Technical University Berlin

Department of Complex and Distributed IT Systems
Faculty IV
Telecommunications Tower
Ernst-Reuter-Platz 7
10587 Berlin
http://www.cit.tu-berlin.de

Master Thesis

Implementation and Evaluation of AI-Based Self-Recovery Approaches in Cloud Systems

Syed Muhammad Hani

Matriculation Number: 389834
20.08.2019

Supervised By
Prof. Dr. habil. Odej Kao
Prof. Dr. rer. nat. Sabine Glesner

Advisor
Alexander Acker

CIT TU Berlin

This thesis originated in cooperation with the Department of Complex and Distributed IT Systems (CIT).

I would like to thank Prof. Dr. habil. Odej Kao and Alexander Acker at the CIT department for giving me the opportunity to carry out state of the art research in this field.

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resource listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, 20.08.2019

.....

Signature

Abstract

Expectations from cloud systems are increasing and traditional concepts revolving around IT service management may not be able to solve upcoming challenges. Cloud systems are expected to handle incidents and recover from anomalies more quickly and improve key metrics such as mean time to repair and quality of service. Therefore, efforts are being made to equip cloud systems with self-healing and self-recovery capabilities. Self healing cloud systems relies on auto-remediation actions. These remedial actions run on cloud systems using an automated event driven approach. Although automated remedial actions can easily replace tedious IT operations tasks for recovering a system from anomalies, the selection of a remedial action should be cautiously done as to not disturb the overall system state.

This thesis presents the design and implementation of a recovery engine with decision making approaches and connects it with an event-driven automation system. In the first part an event-driven automation system based on the if-this-then-that principle is implemented. A catalogue with recovery actions and workflows is setup using the event-driven automation system.

Experiments are setup on a video on demand service running on OpenStack. For these experiments, anomalies are simulated on the system and results are collected by executing different recovery actions and workflows.

In the second part different AI-based decision making approaches are implemented based on the data from experiments. Given the insight of occurring anomaly, the decision making approach selects the most optimum recovery action from the recovery actions catalog. The implemented decision making approaches consider the impact on quality of service and other workflow related statistics before selecting a recovery option. The recovery engine will require human-intervention only to parameterize decision models at certain touch-points for optimization. In the end, results from different decision making approaches are presented and evaluated.

Zusammenfassung

Die Erwartungen an Cloud-Systeme steigen und traditionelle Konzepte rund um das IT-Service-Managem ent können die anstehenden Herausforderungen möglicherweise nicht lösen. Von Cloud-Systemen wird erwartet, dass sie Vorfälle schneller bewältigen und sich von Anomalien erholen und wichtige Kennzahlen wie die mittlere Reparaturdauer und die Servicequalität verbessern. Daher werden Anstrengungen unternommen, um Cloud-Systeme mit Selbstheilungs- und Selbstwiederherstellungsfunktionen auszustatten. Selbstheilende Cloud-Systeme basieren auf automatischen Korrekturmaßnahmen. Diese Abhilfemaßnahmen werden auf Cloud-Systemen mit einem automatisierten, ereignisgesteuerten Ansatz durchgeführt. Obwohl automatisierte Abhilfemaßnahmen lästige IT-Betriebsaufgaben zur Wiederherstellung eines Systems nach Anomalien leicht ersetzen können, sollte die Auswahl einer Abhilfemaßnahme vorsichtig getroffen werden, um den allgemeinen Systemzustand nicht zu stören.

Diese Arbeit stellt das Design und die Implementierung einer Recovery Engine mit Entscheidungsansätzen vor und verbindet sie mit einem ereignisgesteuerten Automatisierungssystem. Im ersten Teil wird ein ereignisgesteuertes Automatisierungssystem basierend auf dem if-this-then-that-Prinzip implementiert. Ein Katalog mit Wiederherstellungsaktionen und Workflows wird mit dem ereignisgesteuerten Automatisierungssystem erstellt.

Die Experimente werden auf einem Video-on-Demand-Service unter OpenStack durchgeführt. Für diese Experimente werden Anomalien auf dem System simuliert und die Ergebnisse durch Ausführen verschiedener Wiederherstellungsaktionen und Workflows gesammelt.

Im zweiten Teil werden verschiedene KI-basierte Entscheidungsansätze auf der Grundlage der Daten aus Experimenten implementiert. Angesichts der Erkenntnis, dass eine Anomalie vorliegt, wählt der Entscheidungsansatz die optimale Wiederherstellungsmaßnahme aus dem Katalog der Wiederherstellungsmaßnahmen aus. Die implementierten Entscheidungsansätze berücksichtigen die Auswirkungen auf die Servicequalität und andere workflowbezogene Statistiken, bevor sie eine Wiederherstellungsoption auswählen. Die Wiederherstellungsmaschine benötigt menschliche Eingriffe, um Entscheidungsmodelle an bestimmten Berührungspunkten zur Optimierung zu parametrisieren. Am Ende werden Ergebnisse aus verschiedenen Entscheidungsansätzen vorgestellt und bewertet.

Contents

List of Figures

List of Tables

1 Introduction

Cloud systems have significantly evolved over the last decade. Cloud service providers are offerings enhanced tools, technologies and introducing new paradigms that are changing our understanding of cloud systems. The current trends in Industrial internet of things, Fog Computing, Big Data and Artificial Intelligence are driving big changes in cloud and IT operations [128][129].

Cloud systems are becoming more self aware and autonomous [127]. Although there is no defined road map, an eligible candidate towards autonomous cloud system is a self-healing IT infrastructure. In this paradigm, concepts such as Infrastructure as Code [57], Service Containerization [52] and Event-driven automation [64] are likely to play an important role. A Self-healing cloud infrastructure should be able to discover, diagnose and remediate errors. The main purpose of using this approach is to maximize the cloud system's capability on delivering the promise of availability, reliability and scalability [126]. Self healing cloud systems relies on auto-remediation actions. Remedial actions in a cloud environment must be selected very cautiously to not disrupt the operational processes. Remedial actions are not limited to simply restart-on-error. They can include restarting services, scaling up or redirecting packets.The following diagram 1.1 presents a higher level concept of the self-healing approaches implemented in this thesis.

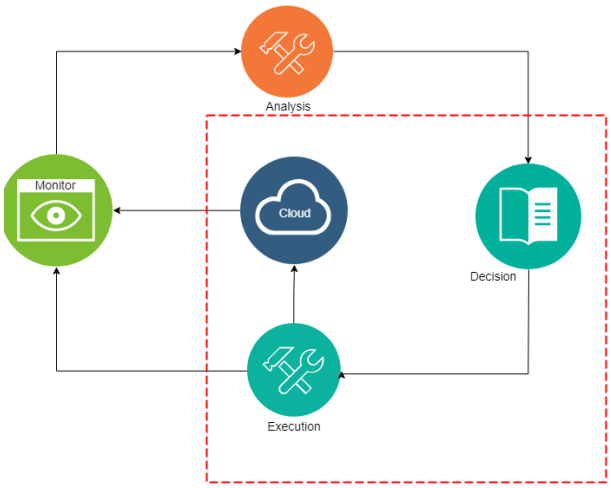

Figure 1.1: Project Scope

The focus for this thesis is in the red highlighted region of the above diagram 1.1. The main task of this thesis was to design and implement a decision making procedure and connect it to a event-driven automation system. A video streaming service running on OpenStack;CIT department's community cloud at TU-Berlin, was used as the testbed. In the scope of this thesis, the Experiment Controller 4.4 includes the execution part and the decision part from the diagram 1.1. It simulates anomalies, runs self-recovery workflows by interacting with OpenStack and Stackstorm, log results and builds AI-based decision models. Due to the focus on decision and execution, the monitoring and analysis of cloud system is based on the assumption of having detailed knowledge about the anomalies occurring in the system.

1.1 Motivation

Expectations from cloud systems are continuously evolving. Previously IT operations had the sole responsibility of delivering on promises agreed upon in service level agreements(SLA). But with the massive influx of users towards cloud systems, this has become a near impossible task to handle manually. Therefore cloud service providers(CSP) now offer automation in areas such as service provisioning, orchestration and scaling. The goal of bringing automation and self-healing approaches to cloud systems is to comply with SLAs, satisfy customers and eliminate the need for 24/7 on-call staff to remediate error situations. Although automated remedial actions can easily replace tedious IT operations tasks for recovering system from anomalies, the selection of remedial action should be cautiously done as to not disturb the overall system state. This motivated me to research, implement and investigate the approaches to self-recovery in cloud systems.

1.2 Objective

The objective of this thesis is to implement an event-driven automation system couple with a decision making system to select the optimum recovery workflow. Given a number of criterion and recovery workflows, the decision making system should opt for workflow with the least overhead. Different system related statistics such as mean time to repair (MTTR), Quality of Service and Success Rate will help decision making system in choosing the most viable recovery workflow for execution in the given situation. The thesis investigates ways to increase the generality and applicability of discussed decision models by considering cases where decision inputs can be parameterized and prioritized by human experts and other methods where human-intervention is not required.

1.3 Outline

This thesis is separated into 7 chapters. The following section highlights these chapters

Background (2) describe relevant technologies, concepts and paradigms.It gives necessary background for the thesis, in order to understand the problem, motivation and contribution.

Contribution (3) describes the self recovery approaches researched and implemented in this thesis. The need and value of using these self recovery approaches in the context of cloud systems is discussed.

Results (4) describes the experimental setup, system components, used tools, presents findings and results of implementing the self-recovery approaches.

State of the art (5) lists the related works and present state of research and applied solutions concerning the different aspects relevant to the thesis.

Conclusion (6) outlines what I learned through the thesis.

2 Background

In this chapter, I have discussed the necessary background required for the understanding of the thesis. In the first section the role of IT operations along with evolution in this field is discussed. Next, cloud computing is introduced with OpenStack, as it is the testbed for my experiments. Different concepts and paradigms related to software delivery are discussed in the next sections and finally an understanding of event driven automation and artificial intelligence in operations is established.

2.1 IT Operations

Information technology operations or IT operations are the set of processes and services that are formed and utilized by the IT staff of an organization for internal and external clients and also for their own working [12]. The term IT operations broadly encompasses the need to run all IT systems for a successful business. The availability of IT operations department in every organization ensures the working of all operations around the clock for fulfilling business needs.

Almost every organization today have computers and rely on IT operations to cover internal and external needs. However in the context of a software development organization the role of IT operations is often extended and overlaps with other departments, covering everything except for management and software development [12]. IT operations help businesses define and maintain quality of service of their software and hardware products, ensures consistency and reliability, manage services and may aid in provisioning and deployment of servers and services.

The IT operations process prescribed by the Disciplined Agile Toolkit promotes an adaptive, context-sensitive strategy. Disciplined Agile[14] promotes the goal-driven approach for IT operations whereby to reach a decision point you need to consider a range of techniques with pros and cons for each [15]. The following diagram 2.1 describes the IT Operations processes in detail with tasks including running solutions, managing infrastructure and configurations and disaster recovery [15].

Each of these decision points are evolving with the evolving trends and technologies in information technology.

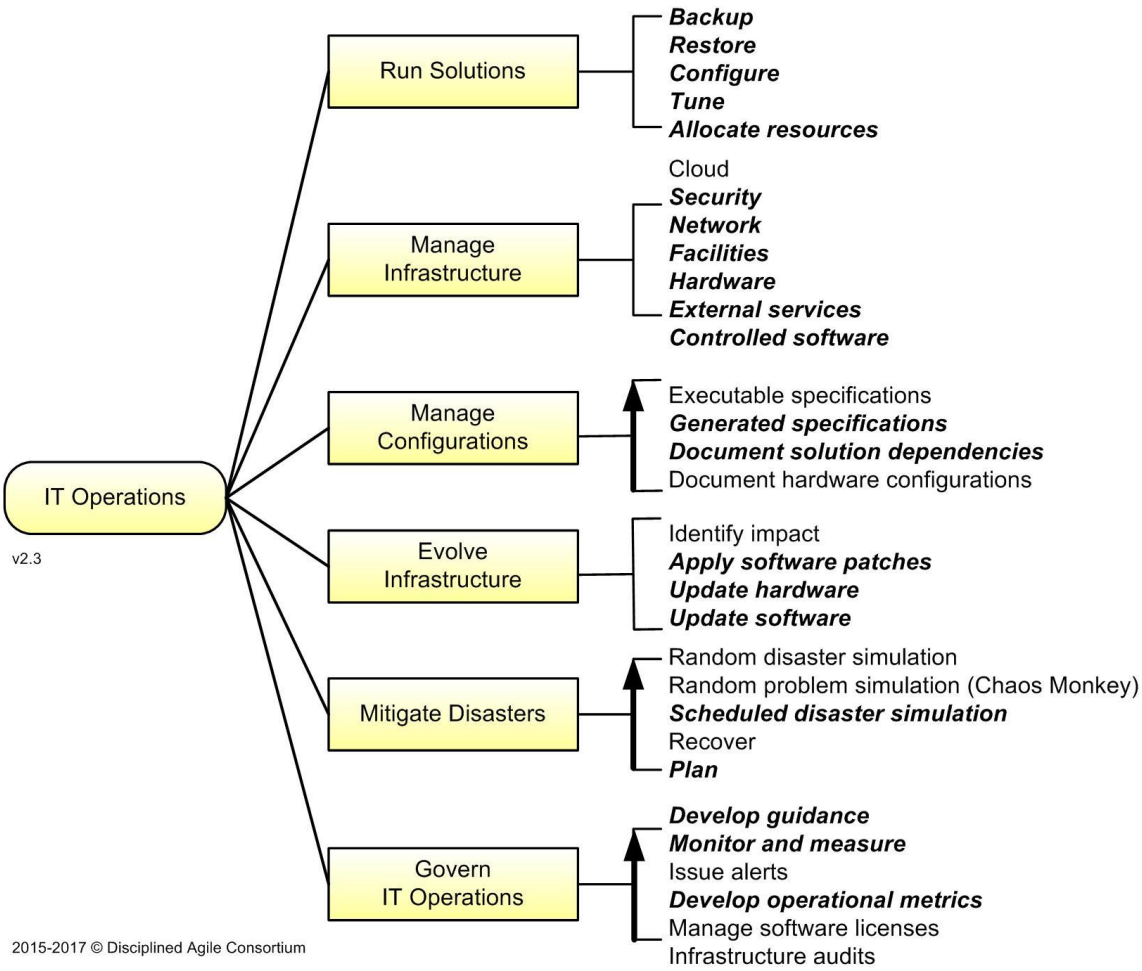

Figure 2.1: IT Operations Process [15]

2.1.1 Evolution of IT Operations

The current technology trends in machine learning, cloud computing, fog computing and industrial internet of things (IoT) are making changes in the area of IT operations [12]. IT operations have to plan for consistency, reliability, scalability and performance of applications. They need to make decisions taking in account, cost and performance matrix and are therefore more involved with the business and management of an organization. The decision of on-premise or off-premise, internal, external or 3rd party and the data/security issues have to be dealt with and planned by IT operations.

2.1.2 Automation in IT operations

IT operation engineers spend a lot of time with diverse tasks ranging from administration and management of both the core infrastructure and the business application state, routine monitoring, tuning, resource management, maintenance and troubleshooting. Therefore the time management and efficiency is the key to better IT operations which can be catered by bringing automation [16]. Automation in monitoring, maintenance and regular changes is prone to less human errors. These automation tools and techniques have been around for a while, with both automating minor tasks using existing tools or involving third party tools, but their adoption has never been easy. This is due to the continuous engagement of operations team in keeping the systems alive which leaves them with less time to test automation and calculate the risks and benefits for adopting a tool or technology.

2.1.3 Site Reliability Engineer

Site Reliability Engineer(SRE) is a working domain created by Google in 2003 when their servers, networks and data-centers grew in size and they required more reliable and efficient approaches to manage it [17]. Following the internet giant, other IT companies also started hiring for SREs to manage their large clusters of servers [17]. Very soon Site Reliability Engineer developed as a completely new IT domain withing IT operations which aimed at pushing automated solutions for operations domain. In an interview, Ben Traynor, VP of engineering at Google and founder of Google SRE, highlighted the role of a site reliability engineer [18]

> *SRE is fundamentally doing work that has historically been done by an operations team, but using engineers with software expertise and banking on the fact that these engineers are inherently both predisposed to, and have the ability to, substitute automation for human labor. In general, an SRE team is responsible for availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning.*

The increasing demand of SRE over traditional IT operation engineers is increasing with the increased usage of cloud technologies and more cloud native applications are pushed in the market with a heavy focus on automation.[19] Along with SRE, other similar working domains include infrastructure engineer, deployment engineer, disaster recovery tester and other roles with different names but overlapping jobs.

2.2 Cloud Computing

This section defines Cloud Computing and discusses its impact on IT operations. Understanding the requirements, models and characteristics of Cloud Computing is essential

to create bases for this thesis.

NIST [1] defines Cloud Computing as a rented shared pool of resources that are ubiquitous in nature and can be accessed on-demand. The pool of network, application, storage and server resources in a cloud computing environment are easy to configure, provision and release. Cloud Computing is also considered computing as a utility [2]. Cloud computing took the server and storage planning and requirement concerns from software developers so that they can focus on making scaleable applications and generating profits. NIST [1] lists five characteristics required by a cloud system. These characteristics are essential and can be considered as requirements to be qualified as a cloud system. From a business point of view, there are three service models offered by a cloud provider and four deployment models [1].
According to NIST [1] there are five requirements for cloud computing:

- A consumer can select resources to provision, manage applications, network and storage capabilities without a human help. The on-demand capability of a cloud system is ensured by a permanent availability of cloud resources.
- All the capabilities given by a cloud service must be accessible over a network via a standardized mechanism so that heterogeneous client systems can access it. Client system can vary in both software technologies (Java,Python etc program interface) to different hardware; mobile phones, tablets and personal computers.
- Pool of configure-able cloud computing resources should be offered to the consumer as a multi-tenant model whereby hardware resources are virtualized to provide on-demand access to consumers without giving out any details of the location or capacity of hardware resources in use. Consumer may be able to specify a location at a higher abstraction layer for e.g. selecting the location of data-center. This quality gives the illusion of clouds' unlimited available resources.
- Provided resources are elastic in nature and can be provisioned or released. In some cases this can be triggered automatically by setting thresholds or limits for auto-scaling. The elasticity should adjust to current load requirements of the application. Cloud systems monitor its resource (storage, network, processing etc) utilization and optimizes at a certain abstraction layer.
- Cloud computing offers measured services which enables pay-as-you-go models. This requirement is mainly associated with billing

There are three service models for cloud computing as described by NIST [1]. They are as follows:

- Software as a Service(Saas)
  Software as a service model offers users to use provider's applications running on a cloud platform. In this model, the user is unaware and has no control over the underlying infrastructure on which the application is running. Only a limited amount of settings can be configured by the user.

- Platform as a Service(PaaS)
  Platform as a service model allows users to deploy their applications on the provider's cloud infrastructure. These applications can be consumer created or developed by using programming languages, libraries and services supported by the cloud provider. Compared to SaaS, this model gives more configuration settings and control to the user in terms of application management and deployment but the user cannot control the underlying infrastructure (OS, servers, network and storage).

- Infrastructure as a Service(IaaS)
  Infrastructure as a Service model includes deploying software on cloud platform with the autonomy to provision infrastructure resources; processing, storage, network and other fundamental computing resources. The pool of infrastructure hardware resources is still inaccessible but a greater control on deployed software, storage, network and operating system is granted to the user.

The deployment models for cloud computing described by NIST [1] are as follows:

- Public Cloud
  Public cloud is deployed on the premise of the cloud provider and is accessible to the general public.

- Community Cloud
  Community cloud is deployed for users of a particular community with shared mission, goals or interests. One or multiple organizations can be part of a community cloud. Therefore Community Cloud can be on or off premise of any of the member organizations.

- Hybrid Cloud
  Hybrid cloud is a combination of two or more cloud deployment models.

- Private Cloud
  Private Cloud is provisioned by an organization to be solely used by the members of that particular organization. This type of cloud deployment model can be managed on-premise or out-sourced to a third party to be managed off-premise.

2.2.1 Architecture

A better way to understand the cloud computing space is to visualize it's architecture. The cloud computing architecture proposed by Zhang et al. [3] categorizes it in four layers namely hardware, infrastructure, platform and application. The architecture also highlights the three service models with examples and helps us understand the user privileges on each layer. The first layer i.e. hardware layer represents the physical resources of a system. Infrastructure layer is the virtualization layer over the physical resources which manages the user requirements by instantiating and stopping virtual machines(VMs).

Platform layers defines standards and APIs which help user to communicate with cloud systems and deploy applications and finally application layer contains the user applications and manages requests from applications to the underlying infrastructure. Zhang et al. [3] also states that there is not a well-defined curriculum for each layer and their usage often tends to overlap. This cloud computing architecture is visualized by Zhang et al. [3] using the following figure 2.2

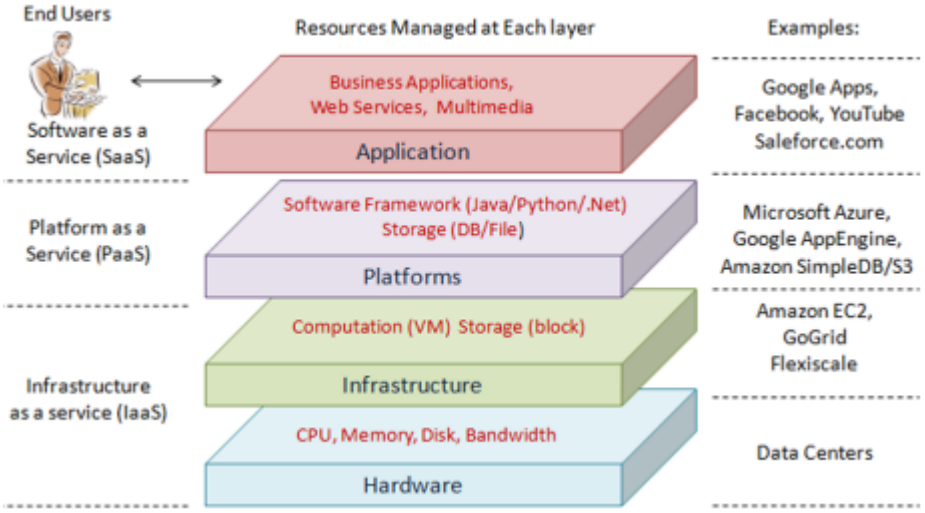

Figure 2.2: Cloud Computing Architecture [3]

2.2.2 Cloud SLA

The aim of IT operation engineers is to keep the customers satisfied by delivering the promises of SLAs. Cloud Service Level Agreement(SLA) is an agreement between cloud provider and user which ensures a certain level is maintained by the provided cloud service [26]. This agreement creates a transparency for the guaranteed service characteristics provided to the client [27]. SLAs can have different attributes based on cloud provider. SLAs can describe attributes such as the agreed scope of services, speed, efficiency, load, billing , response time etc [27].

SLA plays an important role in cloud computing context because organizations need a legal binding service agreement in place when dealing with external cloud service providers [28]. SLAs are formulated considering customer's requirements and their context of service usage [28]. In a cloud computing environment where virtualization is the key, resource allocation, distribution and usage are main factors of a SLA [28]

Every CSP's SLA is different, but there are some concepts that should always be included [29]. Solutions Review [29] highlight five important topics that should be part of every SLA:

- Availability based on Quality of service
- Data ownership
- Cloud hardware and support
- Disaster recovery and backup
- Customer responsibilities

2.2.3 Open Stack

In this subsection, OpenStack[1]; a concrete implementation of cloud computing system is discussed. Openstack is a free software and a IaaS solution to control, monitor and provision pool of configurable networking resources and cloud computing architecture throughout a datacenter. OpenStack offers a GUI dashboard for management and user interaction. These resources can also be accessed and managed by a provided OpenStack API[4]. An active and thriving community of OpenStack developers helps build, maintain and upgrade the software. OpenStack is written in Python programming language and distributed under the Apache 2 license.

The main features of Openstack are defined [6] are as follows:

- Scalable
  Openstack solution is deployed in companies that are working on petabytes of data volume and on massively distributed architectures with billions of stored objects.
- Compatible and Flexible
  Openstack is compatible with many of the virtualization solutions available in the market. Some prominent examples are Hyper-V, XEN and KVM. Its important to note that this is a rapidly changing space where Openstack community of developers work hard to bring compatibility.
- Open
  Openstack, being an open source technology bags all the benefits like code manipulation and modification. For organizations, Openstack comes with prepackaged standards for adaptation making it convenient for companies of different scales to work with it.

[1]OpenStack, https://www.https://www.openstack.org//,Accessed=19-08-2019

OpenStack architecture is conceptualized using the following diagram 2.3. Architecture of Openstack allows users to choose from a variety of necessary or complimentary services in order to meet the requirements of computing, storage and networking. In the following diagram 2.3 all the native components and their interactions with each other in an Openstack environment are shown. These were developed over time by individuals and companies supporting Openstack [7]. This architecture is subject to change as new features are routinely added and removed from Openstack components.

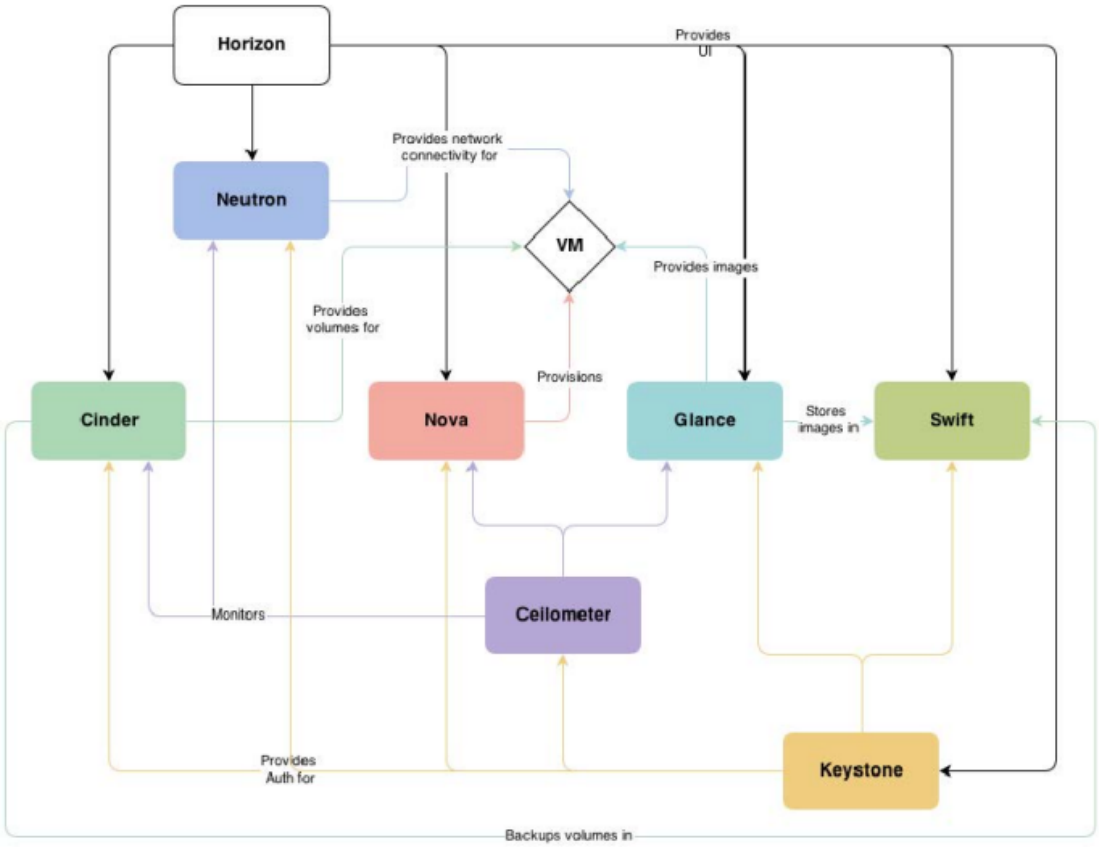

Figure 2.3: OpenStack Conceptual Architecture [7]

Components can be installed or removed from OpenStack installation based on user requirements. Tiago Rosada and Jorge Bernardino [7] categorize OpenStack components in broadly five groups namely:

- Computing
  OpenStack compute and Image service

- Networking
  OpenStack Networking service

- Storing
  OpenStack Object Storage and Block Storage
- Shared Services
  For monitoring, dash-boarding, integrating OpenStack policy and other services
- Supporting Services
  Message queues and other supporting services

As it can be seen from the Openstack conceptual architectural diagram 2.3, there are numerous core and additional components of OpenStack. Therefore I have only described the ones here which are either directly part of the implementation or somehow contribute to the results in my experiments.

- Keystone [10] provides authentication and service discovery by implementing OpenStack Identity API [9]. Keystone service and API allows access to use OpenStack functionalists and build applications that interact with OpenStack.
- Horizon is the UI service providing web based user interface to OpenStack which allows to interact with other services like Nova, Keystone and Swift.
- Nova is the provisioning service of OpenStack which allows to provision compute instances or virtual servers. Nova requires a set of OpenStack services for its functioning, Keystone (identity service), Glance (image repository for compute instances), Neutron (connects to virtual or physical networks after) and Placement (service for tracking inventory of a virtual machine)[11]. Nova service can be used by either Nova Client, OpenStack API or Horizon.
- Heat is the orchestration service of OpenStack which uses a Heat template describing a cloud orchestration. Internally it uses the OpenStack native API to make calls to orchestrate cloud applications.
- OpenStack native REST API launches server instances, create, stop, run and build images and containers and completes other actions. Like other cloud providers Open Stack also offers a native Python API bindings. It simplifies the process of writing automated Open Stack scripts for efficient administration and takes the load by automating manual administration tasks [56].

2.3 Containerization

As discussed in Cloud Computing 2.2 section, virtualization is an integral part of a cloud system. In this section I have discussed a evolved approach to virtualization. Containers; a modern approach to virtualization, is changing the way software is developed and distributed [52]. They have the capability to transform IT operations and cloud management. Containers make developers run out of the excuse *but it works on my machine!* [53] and allow them to build software locally which is able to run on any

hardware stack whether it be in an IT department, on a personal laptop or off-premise using cloud technology [52]. Containers are saving a lot of time of operation teams as they no longer have to work tirelessly to configure environments and manage software and hardware dependencies [52]. Containers guarantee resource isolation; a property which ensures that a user application does an impact in any way another application by the same user [51].

OS-level Virtualization or more commonly known as Container-based virtualization uses OS resource isolation properties to create an operating system abstraction [51]. Container-based Virtualization creates containers; uniquely identifiable user spaces, which share the host system network, storage, compute, memory resources and share a single OS kernel [51]. Containers can also be defined as isolated guest virtual machines running on OS kernel [50]. Similar to a virtual machine, containers encapsulates all application dependencies [52]. Although appearing to be quite similar to a virtual machine, containers bags certain advantages which are possible using a traditional virtual machine based approach [52]. Following are some of the advantages: [52]

- Container are way more efficient then virtual machines because they share host resources
- Containers can be started or stopped with a click and a boot time of a few seconds
- Efficiency of containers ensures that application running on it incur none to a little performance overhead
- Portability of containers are is better then virtual machines
- Containers are light-weight, allowing developers to simulate a production environment locally by running multiple containers
- Containers saves a lot of software developer's time by packaging complex applications. They need not to worry about configurations and the impact of changes they make to their system on running containerized-applications.
- Software development and testing teams do not have to worry about the differences in environments and dependencies [52]

The concept of Containers as part of a hardware/software system is given in the following diagram 2.4

Scheepers Mathijs Jeroen [49] compare conventional (hypervisor-based) and container-based virtualization. Both techniques provide a degree of resource isolation and try to optimize hardware utilization across hardware resources. They [49] argue in their research that both the techniques have different benefits based on the use case. If equal resource distribution is required then hypervisor-based virtualization should be proffered. On the other hand, if a person wants to execute multiple isolated processes while

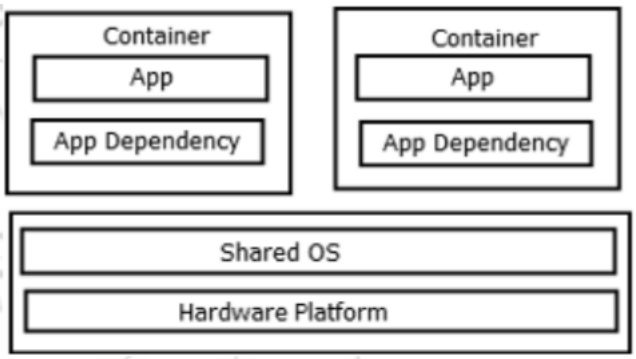

Figure 2.4: Containerization [50]

trying to make the most out of hardware resources then container-based virtualization is a better method.

The industry standard for containerized application is Docker[2]. Containers are not a new concept. UNIX systems (systems with a unix based shell) have had the chroot command which isolated the UNIX filesystem [52]. Containerization evolved over the years and finally in 2013 Docker entered the technology and virtualization market with the promise of bringing the final pieces of the puzzle together [52]. Since then, in just six years, this technology have witnessed tremendous support from the IT industry and now it has began to enter the mainsteam [52]. Docker is offered in the form of Docker container images; a lightweight, executable and isolated package of software and its dependencies. Docker container images are shipped with the following, covering all the requirements to run a application [52]:

- Application Code
- Runtime
- System tools
- System Libraries
- Other required settings

Docker is available for both Widows and UNIX based operating systems. Docker image is always executed and run the same way regardless of the platform and infrastructure [53]. Docker Images running the Docker Engine bags the following properties [53]:

- Docker is the industry standard for containers
- High portability

[2]Docker, https://www.docker.com/,Accessed=19-08-2019

- Docker containers are lightweight which reduces server and licensing costs
- Isolation ensures secure applications]which are running in Docker containers

Docker containers as part of the IT infrastructure can be visualized using the following diagram 2.5 Docker Images run on Docker Engine. Docker Engine is the standard

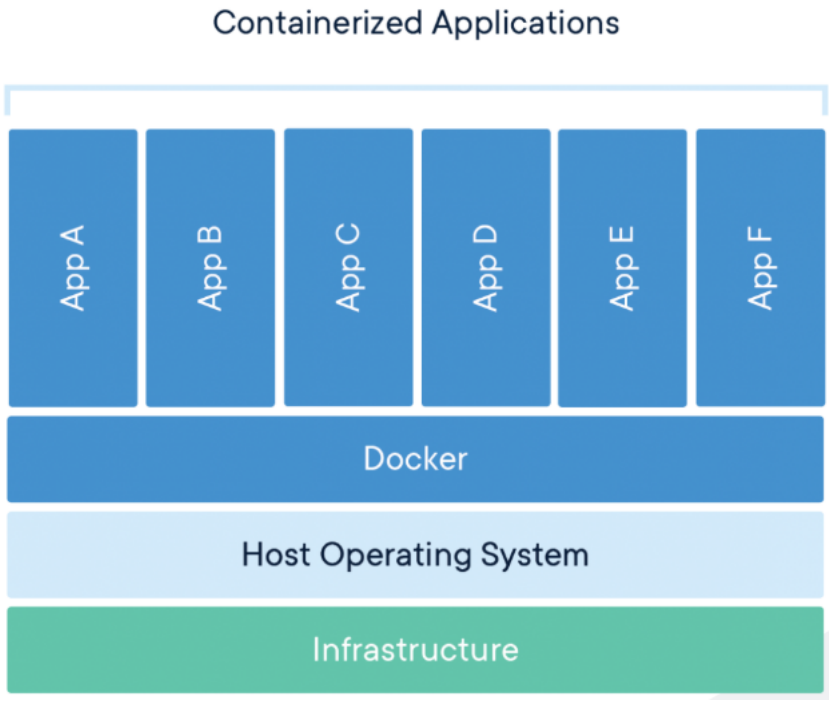

Figure 2.5: Docker[53]

container runtime shipped by Docker [53] that can run on both Windows and various Linux distributions including but not limited to CentOS, Debian, Fedora, Oracle Linux, RHEL, SUSE and Ubuntu. Docker's unique tooling and packaging approach parcels all application dependencies within a container which can then be easily run on the Docker Engine. Docker Engine enables Docker containerized application to run anywhere [53]. The user interface provided by Docker Engine is easy to use. Docker made it easy for everyone to spin up an application using few easy commands [52]. Some of the key offerings by Docker are [52]:

- Docker Hub is the repository of Docker images from where a user can easily and quickly get started by pulling and running images. Docker Hub also helps avoid duplication
- Docker Swarm is the cluster manager for running Docker containers
- Kinematic is the graphical user interface to work with containers
- Machine is the command line utility used for provisioning of Docker resources

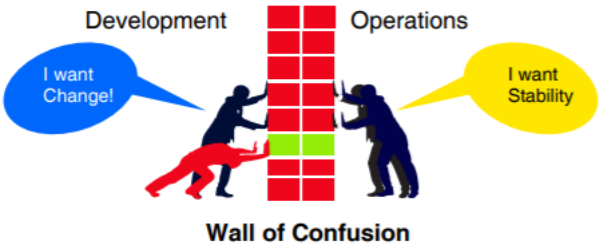

Figure 2.6: Operations vs Development[25]

2.4 DevOps

The development in IT operations and Cloud Computing lead to the foundation of a new paradigm. This section highlights the DevOps paradigm in context of evolved IT operations and Cloud Computing. Traditionally the work of IT operations and software development have been done in separate silos. In the previous sections, I have discussed that this distribution leads to a number of misunderstandings and conflict. Considering IT operations and development as separate units can lead to many problems. Experts suggest that these problems can bring negative effect on the working and performance of an organization [20]. The following figure 2.6 shows the conflicting ideas of development and operations.

DevOps have emerged as a new paradigm in the recent years aimed at bridging this gap between the units of developers and organizational personnel [20]. DevOps is a combination of *Dev* for software development and *Ops* for IT operation but defining what a person entitled as *DevOps Engineer* do is still ambiguous and open to interpretations because DevOps is more of a culture then a straight forward framework [24].

The definition and understanding of DevOps varies across organizations, experts and contexts. There is still a vague understanding of the term and organizations working under DevOps umbrella often tends to differ with each other. According to Farroha [21] DevOps is a set of rules and principles designed for operations and development to work together. This integration of units is aimed at maximizing customer satisfaction and increasing service quality [21]. Dyck et al. [22] defines DevOps as

> Organizational approach that stresses empathy and cross-functional collaboration within and between teams - especially development and IT operations - in software development organizations, in order to operate resilient systems and accelerate delivery of changes.

DevOps life-cycle is described using the following conceptual diagram 2.7. The diagram 2.7 shows the working areas and tasks involved in DevOps.
DevOps follows agile methodologies and lean methodologies used in software development life cycle. According to Dyck et al. [22] DevOps is not only limited to operations

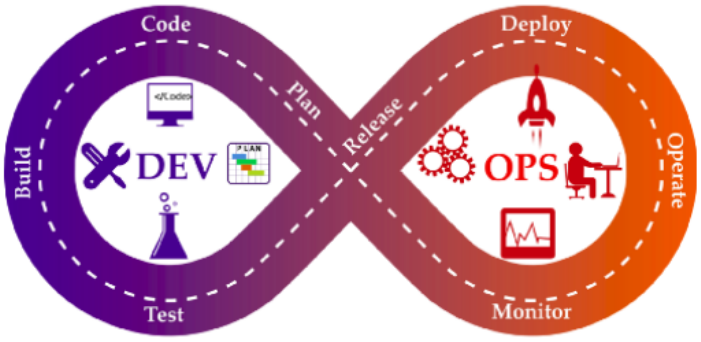

Figure 2.7: DevOps Conceptual Cycle [24]

and development teams, but also involves business managers, software quality insurance personals, business developers, requirement engineers and software analysts. Which makes a lot of sense, since without the involvement of all parties a faster delivery of software is not possible.

There are four main aspects of DevOps[23]:

- Culture
  DevOps culture brings more involvement of operations team into development. Development and Operations team must synchronize their work, tasks and attend each others meetings to understand the direction and pace of work.

- Automation
  Bringing automation to software builds, deployment and testing is an integral part of the DevOps paradigm. Every change in the system run through automated tests via a pipeline which ensures that a modification doesn't bring the whole system down.

- Measurement
  Measurement of key metrics which are shared across teams to deliver key promises to businesses. Performance measurements should be made available to all members across teams that are involved in the software development and delivery process.

- Sharing
  Sharing and collaboration among developers and operations stands out to be one of the main concepts of DevOps. Sharing can be across tools, technologies, managing environments, knowledge and ideas.

DevOps can be implemented in three phases [24]:

1. Automated Testing
2. Continuous Integration
3. Continuous Delivery

The following subsections discuss these three DevOps implementation phases in detail.

2.4.1 Automated Testing

Manual testing; the traditional method of testing involves a human expert creating, maintaining, recording tests and evaluating results on completion [30]. This is a tedious process which require a lot of physical time and effort to ensure that the software works as intended. Evaluation and interpretation of results also require more time since the tester have to check log files, databases errors, system errors, integration errors and application errors and record them manually [30]. Therefore the main motivation to automate tests is to stop repetitive work and save time [30].

Automation testing involves creating a test case or a combination of test cases (test case suite) which replaces manual tests [31]. Apart from creating test cases, the advantage of using automated tests is that if can enter test data into the System Under Test during test execution, compare the actual and given results and generate reports based on the results [31]. A test automation tool can re-execute the tests on demand and can be completely controlled by human testers i.e. start or stop during execution [31].

The Automation Testing Process is given by the following state diagram 2.8:

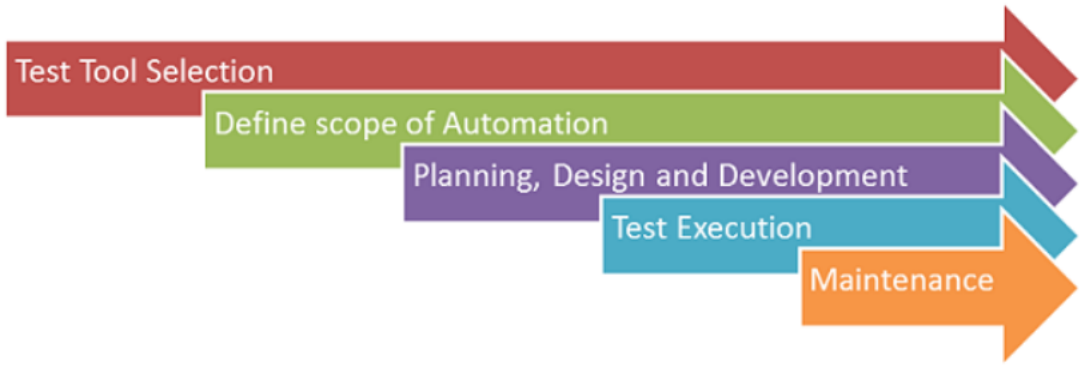

Figure 2.8: Automation Testing Process [31]

2.4.2 Continuous Integration

The process of software development involves breaking down the development in units or individual modules [33]. Usually software development process run across different

teams working individually on one or multiple modules [33]. On completion of the module, changes need to be merged in an integrated software. Traditionally these changes were merged once at the end of every day and complied on a build server [33]. There are numerous issues in this approach, improper integration or an error in code going undetected creates problems afterwards and is difficult to recognize and debug [33]. Even if there are no errors, teams completing on linked modules earlier then others have to wait putting a strain on development costs [33].

Continuous Integration(CI) is a DevOps practice where software developers push and integrate their work more frequently resulting in more integrations per day [34]. This is enabled by automated build systems and automation testing to verify each software build and detect and debug integration errors earlier in the SDLC pipeline before they become a bottleneck [34]. The process of automating build which makes a working program to compile code, update databases and relocate files should ideally be executable by just a single command [36]. Many teams using this strategy have reported fewer integration problems and a faster software development overall [34].

The following figures 2.9 2.10 compare traditional integration and continuous integration:

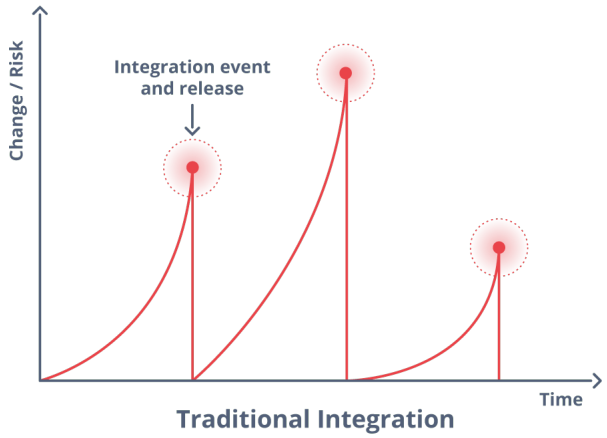

Figure 2.9: Traditional Integration shortcomings [35]

To work with Continuous Integration practice, its preferred that all team members use the same repository or the same source of the working code where all functional components are pooled together [36]. This applies to code and all other fragments required to compile a functioning software including but not limited to databases [36]. Members of software development team working with continuous integration should have access

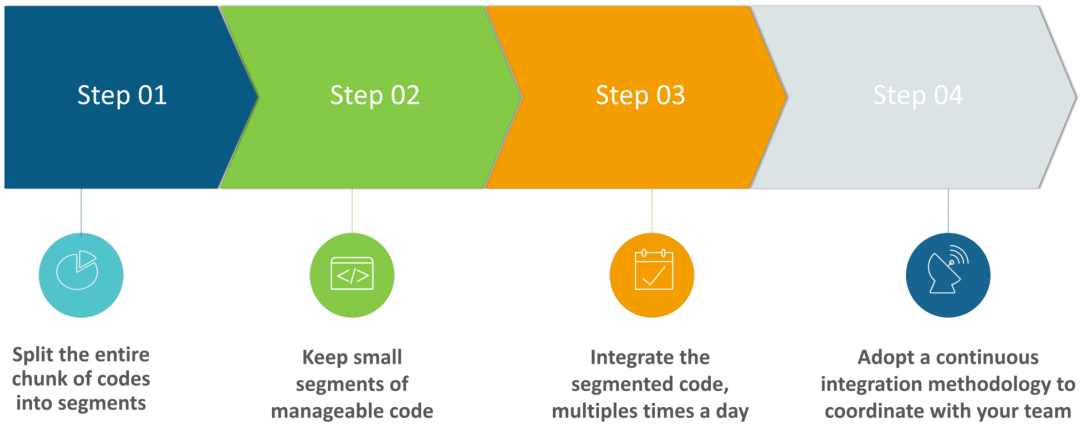

Figure 2.10: Continuous Integration Explained [35]

to the latest version, should be able to clone and work with the copy of the latest version and push their changes in source [36]. Similar to other DevOps practices, tools are not necessary but help enable continuous integration practices. Its important to understand that with a considerable more effort, these practices can be implemented manually without use of external tools [36].

2.4.3 Continuous Deployment

Continuous Deployment(CD) involves all the steps that come after a developer pushes his/her changes in the repository to the automatic release of theses changes to the production phase [37]. This process unburdens the IT Operations team from manual processes of deploying changes [33]. According to Dev Insider [38] continuous deployment and delivery are part of continuous integration. They argue [38] that these methods have been in practice for a long time in software development but got streamlined and well-defined recently.

CI/CD and automated testing as part of the software release pipeline can be visualized using the following figure 2.11

2.4.4 Continuous Delivery

Continuous Delivery is a set of tools, techniques and processes that help in reducing the time of delivery of a software [41]. It allows customers to request and obtain the latest executable version of software which is error free [41]. Software development life cycle typically consists of three steps mainly software development, software quality assurance and software delivery [42]. These were handled in silos by dev,QA and Ops teams. The following figure 2.12 highlights the role of Continuous Delivery from the initial phase of idea to the last step of delivering software to the customer [42]. An organization following Continuous Delivery practices is likely to gain the following benefits [43]:

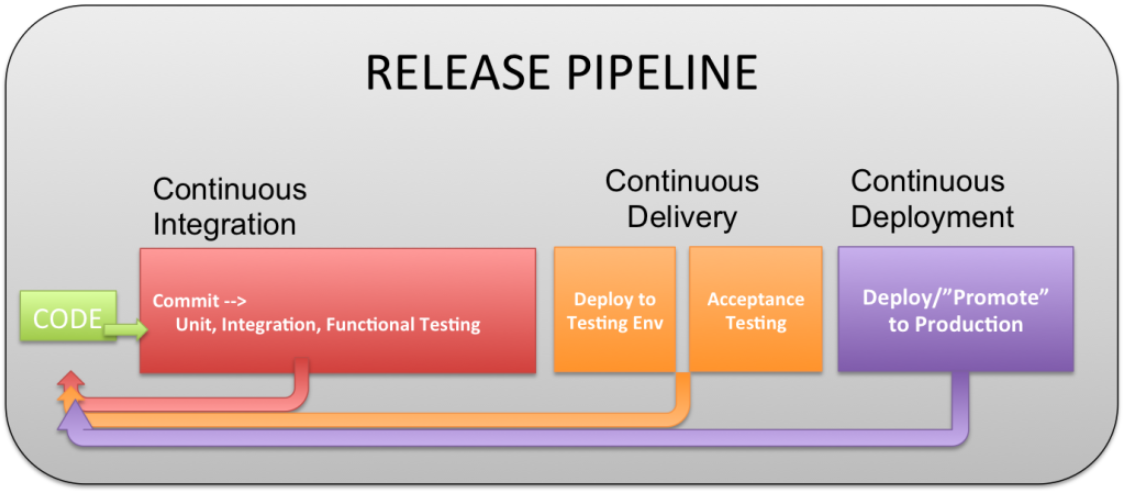

Figure 2.11: Software Release pipeline [39]

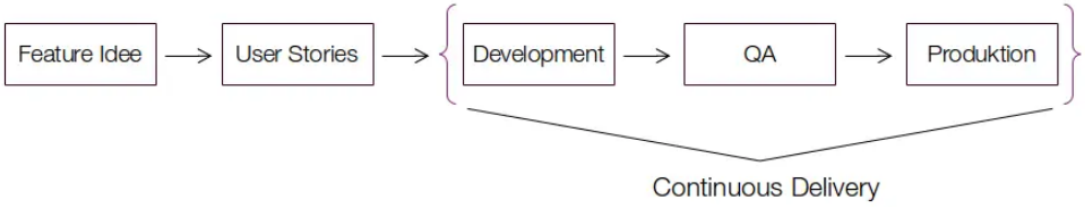

Figure 2.12: Continuous Delivery Chain [42]

- Low risk releases with less chances of errors
- Faster time to deliver software
- Higher product quality
- Lower costs of software delivery

2.4.5 Software Configuration Management

Software Configuration Management is the concept of software versioning. The recent advances in the field of DevOps has improved SCM and increased its working area, but still there is no universal agreed upon definition of term SCM [45]. Dev insider [45], a software magazine, describes SCM as a sets of standards and procedure for software development and software management of a evolving software system. Stackify [47], a US-based company dealing with software related troubleshooting issues, defines SCM as a set of processes, policies and tools that helps to organize the development process

and saves the current (baseline) state of the software and enables developers to work on enhancements, features and fixes. Another SCM definition shared by Software Testing Help [48], a software testing blog, explains SCM practices as something that adds controls to the establishment of software baselines. Software Configuration Management main roles given by IEE Standard 729-1983 [46] are:

- Identification of the product structure across old and new versions
- Controlling software release and modifications
- Recording and reporting for status of all components involved in software development
- Audit the completeness and work done in a version and ensure consistency

Another important terminology in the sphere of DevOps is Infrastructure as code (IaC). Although this is not considered a part of the DevOps implementation pipeline 2.7, IaC is empowering modernization in the IT infrastructure. In the next subsection I have tried to shed some light on IaC concept.

2.4.6 Infrastructure as Code

DevInsider [59] defines Infrastructure as Code (IaC) as the in-house organization, management and provisioning of resources in an environment. According to DevInsider [59] the understanding and definition of the term as code is ambiguous and open to interpretation. The main advantage of having Infrastructure as code it the accessibility of infrastructure to software developers, usually to the software development tools such as version control [59]. Computer Weekly [57] describes IaC as programmable infrastructure which is an enhancement to the manual processes of code deployment used by operation teams. IaC uses high-level descriptive languages to code delivery and deployment procedures [57]. What-is [58] explains that IaC treats infrastructure configuration exactly as a programmable software with the use of tools and easy to understand descriptive languages. With the introduction of IaC, software application can include the orchestration and deployment scripts with the application execution steps and in return bridges development and operation teams closer to each other; a core principle of DevOps [58].

For my thesis work I have used Ansible which works on the principle of IaC. Ansible[3] as defined by their own website [60] is a tool which simplifies IT automation by reducing the amount of repetitive tasks performed across operations, development and DevOps teams. Ansible covers automation tasks including but not limited to cloud provisioning, configuration management, deployments and service orchestration [61]. Ansible, within its scope, orchestrates not just the services but also manages the interconnections and

[3]Ansible, https://www.ansible.org//,Accessed=19-08-2019

interrelations across related systems and services. Ansible runs on its own, without the help of any custom agents and with a fairly simple automation language which is very close to plain English [61].

Key elements of Ansible according to company website [61] are:

- YAML for yet another markup language, is a minimal markup language based on XML [62]
- Automation jobs can be written in YAML in Ansible Playbooks. Playbooks can carefully orchestrates specified parts of your infrastructure with a lot of control
- Modules are small programs which run a desired task. A playbook may consist of multiple modules
- Inventory files specify which machines Ansible have to manage. Inventory files are specified by INI extension and can be used to group server nodes according to their role or desired automation tasks

In the next section I have discussed an essential concept of Event-driven automation which has the potential to modernize IT operations and cloud systems. Event-driven automation can also help in enabling DevOps practices.

2.5 Event-Driven Architecture and Event-Driven Automation

Gartner [67] considers Event-Driven Architecture (EDA) a design paradigm whereby software components work on a request and response bases by receiving and sending events or notifications. EDA is not as tightly-coupled as the famous client/server architecture because in EDA the sender does not need to know the identity of receiver [67]. EDA is also defined as the framework that manages events and their response production, detection and consumption [64]. Instead of a client/server, EDA consists of event creators and event consumers. Creator is often described as the source event who knows about the details of event creation and consumers are units that are informed about the event and will later be involved in processing, running and in some cases generating output or some kind of results by being affected by the event [64].

Gartner [66] highlights some of the trends and examples from the IT businesses that are enabled by the use of EDA:

- Events with situational awareness enable a rich customer experience
- IT operations working on near real-time scenarios to help elevate the business standards
- EDA can be key component to IoT based sensor networks

- EDA enables Artificial Intelligence and Machine Learning since their processes involves a number of event collection, analysis and response.

The following figure 2.13 shows how businesses benefit from event driven architecture

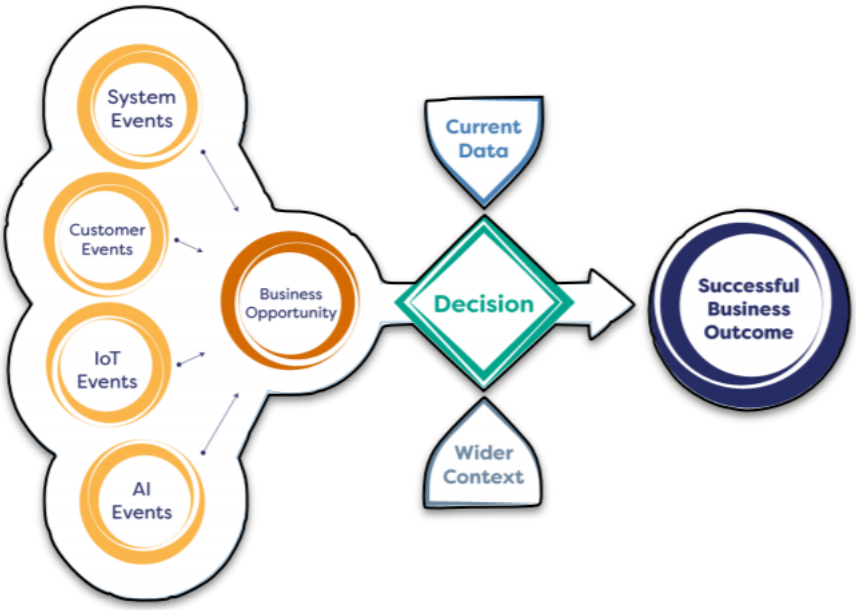

Figure 2.13: Improving Business with EDA [66]

A case study published by Daitan [65] shows how even-driven automation can be a precursor to artificial learning and machine intelligence by funneling useful input data and in return making smart decision. Event-Driven automation is applied by running event driven applications based on event-driven architecture. Such applications have emerged in the areas of ITops, NetworkOps and Security Ops helping solve conflicts between developers, operations and security teams while at the same time full filling cloud SLAs and enabling DevOps practices [68].

In the scope of event-driven automation, If-this-then-that(IFTTT) is a simple paradigm which means that an action is run if an event is detected [69]. In the IT landscape, tools and services using the IFTTT paradigm allows creation of connections and integrations between two or more unrelated services to automate daily tasks [69]. The figure shows 2.14 how IFTTT can be visualized as a simple one way model.

For my thesis, I have used Stackstorm which is a tool implementing event driven automation based on IFTTT principle. Stackstorm[4] defined by the company website [71]

> StackStorm is a powerful open-source automation platform that wires together all of your apps, services and workflows. It's extendable, flexible, and built with love for DevOps and ChatOps.

[4]Stackstorm, https://www.stackstorm.com//,Accessed=19-08-2019

Figure 2.14: IFTTT [76]

Stackstorm is an automation platform that applies to a wide variety of use cases including IoT, cross-system relays, smart-home automation, auto-scaling etc [71]. Stackstorm service is build on IFTTT and follows the core principles of pluggable-rule-execution engine [70]. Stackstorm runs actions or complex workflows by detecting an event or sensing a pushed change. Stackstorm supports SSH-based remote controls, native Ansible playbooks and agent-based tools via integrations [70]. Stackstorm offers a graphical user interface, CLI and REST APIs to interact with its various services and components. These three options are available in both free and commercial versions of the product [70].

Stackstorm main components can be described as follows: [70]

- Sensors
  Sensors are plugins written in Python which senses inbound events or receive events from integrated external systems. In response sensors generate a Stackstorm trigger to be emitted into the system [70]

- Trigger
  Stackstorm represents external events as Triggers. Triggers can be Stackstorm internal or from third party e.g. JIRA issues update etc. Triggers are defined with sensors [70]

- Actions
  Actions are performed by Stackstorm integrations. For instance a spawn VM action can be created using OpenStack integration. Actions can also be custom defined by the user. Actions can be added as scripts, python plugins or by modifying the metadata of already available actions. Actions can be directly invoked using UI, CLI or API, or can be used by calling a rule or workflow [70]

- Rules
  Rules are used to map triggers to actions or work-flows by defining the unique criteria with a set of inputs or payloads.

- Workflows
  In a DevOps environment single actions are not enough to define an IFTTT criteria since most automations are more then one-step [75] Here work-flows come into play by stitching actions together in the order of precedence, by defining transitions and its underlying conditions and passing the required input data. Workflow execution is similar to actions [75]

- Integration Packs
  Integration Packs package the pluggable content available by Stackstorm or third-party vendor integrations which groups triggers, actions, rules and workflows. Since Stackstorm is an open-source project users can create their own projects and push it to Github Stackstorm exchange [70]

- Audit trail
  Audit trail stores context of execution of actions, whether run automated and manually to give results [70]

- Webhooks
  Webhooks are an alternative to Stackstorm sensors allowing integration with external systems using HTTP methods. Sensors use a pull approach to event automation while webhooks use a push approach. Webhooks use HTTP POST request to trigger StackStorm API. Webhooks are ideal to use in service-oriented architecture [75]

- Datastore
  Stackstorm datasource service provide users with the space to store common parameter names and values for resuse in actions, sensors, rules and workflows [74]

Stackstorm software bundle includes a rabbit message queue, mango datastore, graphical webUI enabling creation and configuration of actions, rules and integration packs [70]. Stackstorm internal working architecture is important to understand before creating actions, rules etc. This architecture is explained in detail in the following diagram 2.15

Some of the prominent Stackstorm services are st2, st2sensorcontainer, st2rulesengine, st2timersengine, st2actionrunners, st2garbagecollector, st2auth, st2api and st2stream [73].

The scope of automation with Stackstorm spreads across multiple areas and operational patterns including [72]:

- Facilitated Troubleshooting
  Stackstorm facilitates troubleshooting by stitching monitoring, diagnostics, cloud nodes and communication applications.

- Automated Remediation
  Stackstorm enables auto-remediation by detecting failures on compute nodes, evacuating instances or running remedy actions and if the error is not known alerting system administrators to take further action.

- Continuous Deployment:- Stackstorm connects build and automated tests for continuous deployment scenarios.

Stackstorm Exchange [77] offers integration packs for monitoring services, cloud providers etc. An active community of developers and supports keeps on updating Stackstorm exchange with newer versions, features, fixes and fresh software integrations.

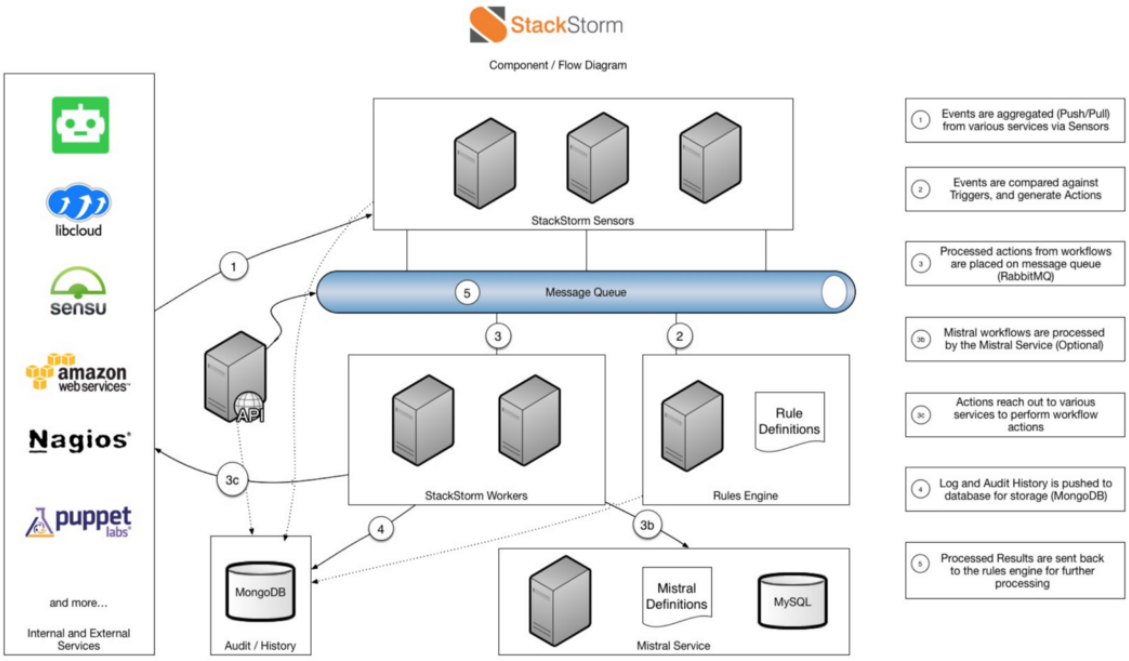

Figure 2.15: Stackstorm Architecture [72]

2.6 AIOps

In this section, I have discussed briefly about the use of artificial intelligence for IT and cloud systems. An Event-driven automation system automates recovery processes and AIOps can bring intelligence to the recovery pipeline.

Artificial Intelligence is a broad term which have been around for decades. Due to the intricacies and depth of the topic, I have only covered a short summary and parts of Artificial Intelligence domain relevant to the thesis. Artificial Intelligence techniques are being applied to the IT Operations and Cloud systems domain. This working area is referred as *AIOps*

Gartner used the terminology AIOps for the first time in 2016. In its Market Guide for AIOps Platforms [80], Gartner describes AIOps platform as a software system that combines Artificial Intelligence and Big data to improve and bring automation to a broad range of IT operations and processes such as performance monitoring, system analysis, event correlation, IT service management and automation. The AIOps platform is visualized as a combination of Machine Learning and Big Data and other concepts by Gartner [80] in the following diagram 2.16

TechTarget, a leading technology blog [78], explains that AIOps make use of the decades of research put into AI for data output, data aggregation, data analytics, automation, algorithms and data visualization and use these technologies for IT Opera-

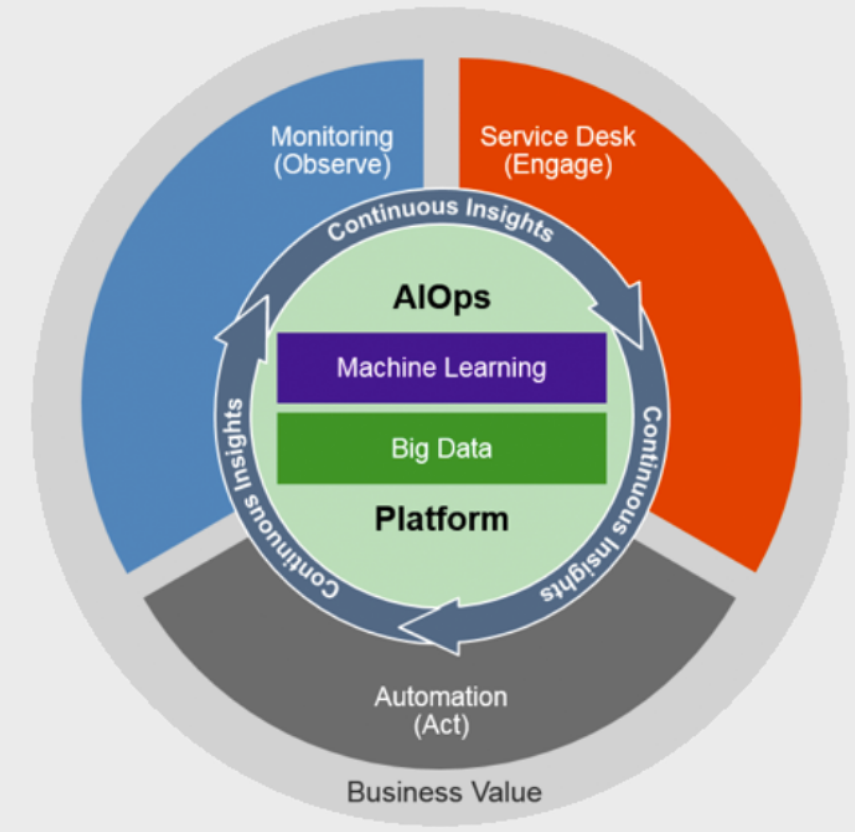

Figure 2.16: Gartner's Visualization of the AI-Ops Platform [80]

tions. Big Data and AI, two of the most transformative technologies of the IT landscape today are put together to ensure IT success. Due to the vastness of data generated by massively distributed operational systems today, the traditional monitoring using domain based tools ends up with skewed insights because of disparate data [79]. AIOps streamlines this data, take data based on some heuristics and in return simplifies and improves the root cause detection mechanism. AIOps is easier to implement by teams already working with DevOps principles since this too breaks traditional working style of working in organizational units and therefore helps innovate [79]

Some of the examples where AIOps is used in organizations are [81]:

- Anomaly or Threat detection
  AI algorithms and heuristics can mine data for potential threats. AI and machine learning models can understand threat patterns and help create a proactive cloud system. AI can help determine root cause of anomalies much faster

- Event Correlation
  IT teams are always faced with tons of long data and alerts. Not all of these alerts

are troublesome. Data mining techniques applied to the continuously incoming log data can determine anomalies by grouping.

- Intelligent Alerting and Escalation
  After understanding anomalies and grouping them by priority, AIOps can act as a routing system, setting remediation actions where human involvement is not required.

- Incident Auto-Remediation
  AIOps can auto-remediate an anomaly by diagnosing the root cause, inferring about the remedial solution and sending alerts to APIs designed for auto-remediation or to IT operation teams.

- Capacity Optimization
  AIOps can optimize the capacity and increase up-time of applications by scaling resources up and down based on requirements.

3 Contribution

In this chapter I have discussed the concepts and methodologies used in the design of event-driven system and decision system for self-recovery approaches in cloud systems. The concepts from the previous chapter 2 are used here to elaborate on the specifics of my contribution and the relation of my work to IT operations, cloud computing and AIOps.

3.1 Design Components

The following diagram 3.1 highlights the design components of a complete self-healing cloud infrastructure

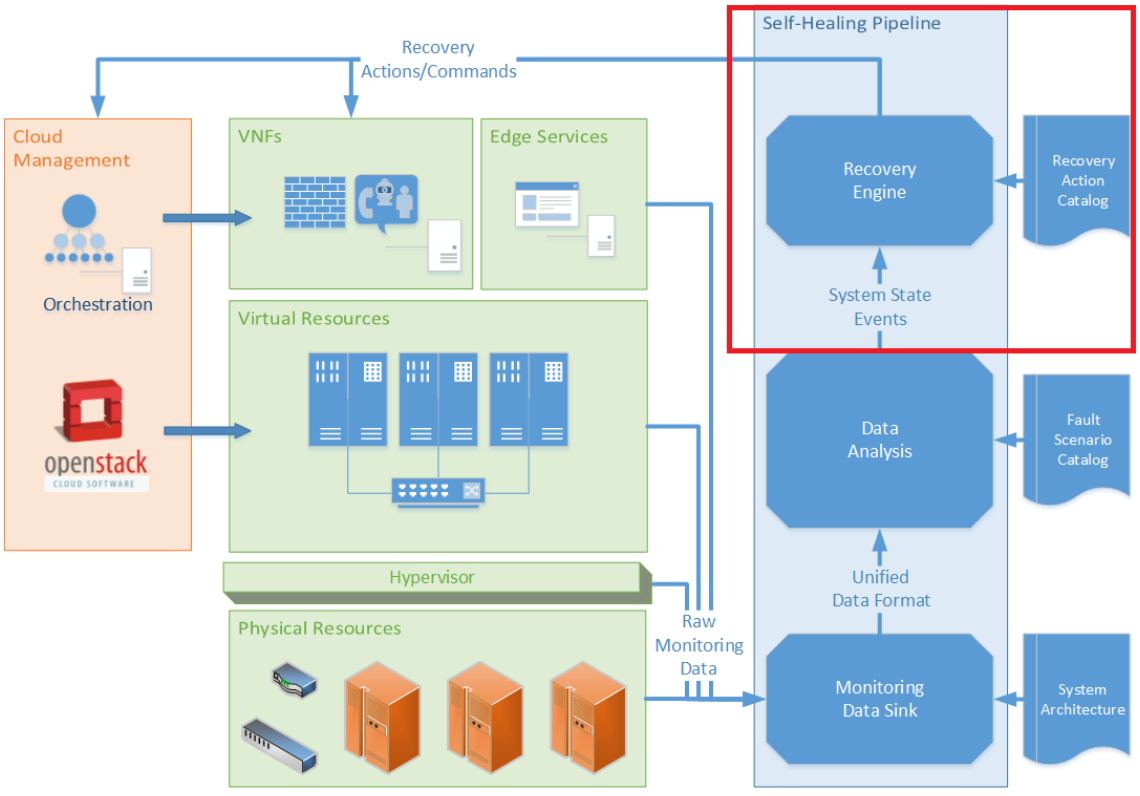

Figure 3.1: Project Scope

As you might have observed in the diagram 3.1, the working principles in the self-

healing pipeline represents principles considered for AIOps 2.6. The focus for this thesis is in the red highlighted region of the above diagram. OpenStack as part of the CIT department at TU-Berlin was used as the CSP. The Experiment Controller discussed in the experimental setup 4.1 receives system state events and maintains the recovery action catalog and along with the decision controller, serves as the recovery engine. This controller simulates anomalies, executes remedial workflows , logs results, builds and executes decision models and evaluates results. Decision models are discussed in this chapter 3 while the details on experiments are discussed in the Results 4 chapter. The underlying assumption here is that the recovery engine receives information about the state of the system(normal, abnormal, etc.) and is aware of the occurring anomaly. Recovery engine is getting this information from monitoring and data analysis components of the self-healing pipeline. VNFs and Edge Services are used here only as an exemplary service deployed on OpenStack cloud and are not covered in the scope of my thesis. For my work, I have used a video streaming service on which recovery actions are executed and evaluated. The following diagram 3.2 explains the components I have used to implement the recovery engine 3.1, recovery action catalog and recovery action/commands as part of the self-healing pipeline.

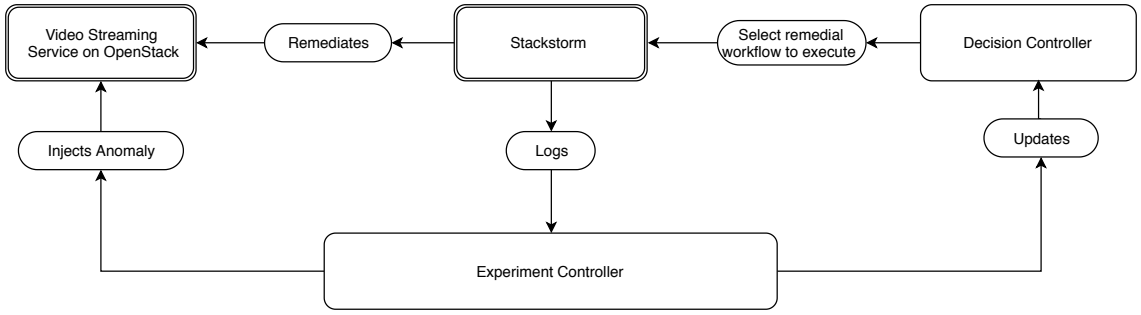

Figure 3.2: Components Flowchart

In the next section I have discussed the concept of Closed Loop Automation which was the guiding principle throughout this thesis.

3.2 Closed Loop Automation

Automation has become an integral part of cloud services, orchestration and IT operations by streamlining ops and delivery. This started with the simplest use of custom case specific automation scripts to fully developed tools providing automation in all spheres of software delivery life cycle [82]. Closed loop automation aims to reduce the need for human intervention in operational processes since its time-consuming, inefficient, boring and with new generation of devices and cloud services, nearly impossible [82]. In a networking context, closed-loop solution is an amalgamation of monitoring, analysis and orchestration [85].

The following diagram 3.3 gives a conceptual understanding of a closed-loop system

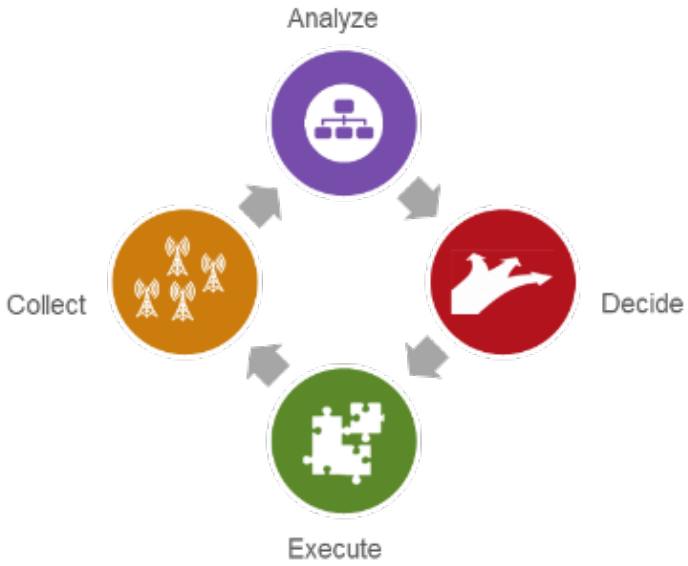

Figure 3.3: Closed Loop Automation [86]

These steps can be briefly described as follows [85]

- Collect and monitor services for operational alerts
- Analyze the alert. Depending on the system it can categorize it, fetch relevant data or create tickets in an incident management system
- Decide which auto-remediation or self-recovery approach to run based on data and some logic
- Execute the self-recovery plan

Closed loop automation is a successor to closed-loop-control systems which have been applied over the decades to a number of power and process engineering use-cases [83] [84]. According to Network World [85] any closed loop-automation system should have the following

- An efficient closed-loop automation system should be easy to scale.
- It should work comfortably in a partner-ecosystem and make use of already available software, service and hardware integrations of the partner-ecosystem.
- It should support both legacy and new infrastructure components.

Self-healing networks and cloud systems are an outcome of a properly designed closed-loop system [85]. The aim of my thesis is to experiment, develop and evaluate control-loop automation as part of the self-healing pipeline. The advantages of closed-loop automation includes better speed and agility, simplified troubleshooting, reduction of human errors, reduced service/application downtime and shift the focus to optimization [85]. Therefore human-intervention is only required for parameterizing the AI-based models at certain touch-points for optimization.

In the next section I have discussed the methodological approaches to self-healing that are implemented in the Decision Controller component 3.2.

3.3 Multi-Criteria Decision Making

Multi-Criteria Decision Making or Multi-Criteria Decision Analysis is the super class of all algorithmic models which are involved in selecting the best alternative from a list of candidates in a decision. Every decision has a multiple factored criteria which is either explicitly defined or appear voluntarily. Based on the criterion and decision alternatives a potential decision is selected. There are numerous MCDM models available that map to both simple and complex problems [87].

In the context of my thesis, the recovery workflow/strategies can be considered as alternatives to choose from given set of criterion. The criterion can be based on system statistics, measurements from running the recovery workflows in the past or human-assigned values. The how-to of mapping MCDM/MCDA for decision making system in our case is discussed with experimentation and implementation in Results chapter 4.

In the next subsections, I have discussed the strategies to implement the decision system using normalized weighted sum model, TOPSIS and entropy weighted model for the selection of optimum recovery workflow.

3.3.1 Weighted Sum Model

Weighted Sum Model or most commonly called Simple Additive Weighting (SAW) establishes a linear value function based on a summation of scores representing the desired goals under multiple criterion multiplied by the specified weights [91]. It is important to note here that Weighted Sum Model is applicable as long as all the data is expressed in exactly the same unit [90].

Even considering the simplest criterion for example, mean time to repair(time expressed as seconds) and quality of service(in frames/sec), its evident that the weighted sum runs into a multi-class dimensionality problem. Inspired from the work of Helff, Florian and Gruenwald, Le and d'Orazio and Laurent [99] I implemented a normalized simple weighted sum model. In this controlled normalization using normalized simple weighted

sum model, every numerical value(Aij) is scaled to a new range of with new minimum and new maximum. The following formula was used to scale old data ranges to new data ranges

```
OldRange = (OldMax - OldMin)
NewRange = (NewMax - NewMin)
NewValue = (((OldValue - OldMin) * NewRange) / OldRange) + NewMin
```

[130]

A weighted sum model is defined by alternatives(m) and decision criterion(n). The values of these criterion are set such that higher the value, higher will be the benefit from that criteria. Weights are relative to the criteria. The following equation 3.1 gives the score of an alternative (Ai) based on criterion (Cj) considering performance value of alternative (Aij) summed with every criteria in consideration: [100]

$$A_i^{WSM-score} = \sum_{j=1}^{n} w_j a_{ij}, for i = 1, 2, 3, \ldots, m. \tag{3.1}$$

The best alternative is the one with the highest value of 3.2

$$A_i^{WSM-score} \tag{3.2}$$

The steps to implement a weighted sum model are as follows:

- Step 1: Design the decision matrix with n criteria and m alternatives
- Step 2: Normalization of Aij values to the standard scale
- Step 3: In this step weights are assigned to each criteria and then multiplied by Aij values in the normalized weighted sum model matrix. Weight values should sum up to a total of 1 [100].
- Step 4: Apply the formula for Awsm-score on the calculated numerical data for all alternatives.

3.3.2 TOPSIS

TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution) is an approach which categories alternatives by distance from the ideal solution and the worst solution in a multi-dimensional space [91]. The goal of using this technique is to determine the relative advantages of alternatives. TOPSIS have been applied to multiple domains, including but not limited to manufacturing engineering, marketing, environment, human resources, logistics, transport and supply chain management [90]. The emphasis of TOPSIS is to find the best possible option which has the highest distance from the worst condition and the smallest distance from the best condition. Different distance functions are applied for this problem [101]. Lets have a look at the steps of TOPSIS implementation and the equations governing it [101]:

- Step 1 - Create a decision matrix
  In the first step a decision matrix is created with alternatives(m) and criterion(n). Intersection of each alternative is x_ij which forms the matrix is given in 3.3

$$(x_{ij})_{m\times n} \tag{3.3}$$

- Step 2 - Normalize
  Second step is to normalize the value of matrix by using the following equation 3.4

$$(x_{ij})_{m\times n}(x_{ij})_{m\times n}. \tag{3.4}$$

  to from rij(normalized values) based on the given formula 3.5:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=1}^{m} x\_kj^2}}, \quad i = 1, 2, \ldots, m, j = 1, 2, \ldots, nr_{ij} = \overline{x_{ij}}\sqrt{\sum_{k=1}^{m} x_{kj}^2} \tag{3.5}$$

- Step 3 - Apply Weights
  In the third step a weighted normalized decision matrix is calculated by using the equations 3.6 3.7

$$t_{ij} = r_{ij} \cdot w_j, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n \tag{3.6}$$

$$w_j = W_j \Big/ \sum_{k=1}^{n} W_k, j = 1, 2, \ldots, n \tag{3.7}$$

- Step 4 - Determine the best and worst solution
  In this step the worst and the best alternative is calculated using maximum and minimum function for each alternative. Using the following equations 3.8 3.9

$$A_w = \{\langle \max(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_-\rangle, \langle \min(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_+\rangle\} \equiv \{t_{wj} \mid j = 1, 2, \ldots, n\}, \tag{3.8}$$

$$A_b = \{\langle \min(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_-\rangle, \langle \max(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_+\rangle\} \equiv \{t_{bj} \mid j = 1, 2, \ldots, n\}, \tag{3.9}$$

- Step 5 - Determine distance from ideal solution
  In this step the L2-distance between the value of criteria to alternative (i) and worst and best condition is calculated. Formula for calculating the distance from best and worst conditions is given by following equation 3.10:

$$d_{iw} = \sqrt{\sum_{j=1}^{n}(t_{ij} - t_{wj})^2}, \quad i = 1, 2, \ldots, m, d_{ib} = \sqrt{\sum_{j=1}^{n}(t_{ij} - t_{bj})^2}, \quad i = 1, 2, \ldots, m \tag{3.10}$$

- Step 6 - Determine relative closeness to ideal solution
  In this step closeness to ideal solution is determined by using the values of Diw and Dib calculated in the last step by applying the formula 3.11:

$$s_{iw} = d_{iw}/(d_{iw} + d_{ib}), \quad 0 \leq s_{iw} \leq 1, \quad i = 1, 2, \ldots, m. \tag{3.11}$$

- Step 7 - Finally, alternatives are arranged by their ranking based on the value of Siw

There are two methods for normalization available for TOPSIS namely sum and vector methods. Sum or Linear normalization was calculated in step 2 while vector normalization can be calculated by replacing the equation in step 2 to the following equation 3.3.2 [131]:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=1}^{m} x_{kj}^2}}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n \tag{3.12}$$

3.3.3 Entropy Weight Method

In systems and information theory Shannon's entropy [123] method is used to determine the degree of disorder. Shannon's Entropy [123] method can be applied to MCDM models for weight calculation. [124]. This method generates weights for criterion based on scientific principles and its capacity to evaluate alternatives and not on their relative importance. This method can be used with multi-criteria decision making problems when decision makers cannot agree on weights for each criteria. The weights here are objective weights.

According to the definition of entropy [124], entropy of every criteria column value (j) is determined by the following equation 3.4:

$$H_j = -\frac{\sum_{i=1}^{m} f_{ij} \ln f_{ij}}{\ln m}, (i = 1,\ldots, m; j = 1,\ldots, n)$$

Figure 3.4: Entropy Weighted Model 1 [124]

Where Fij is determined by the following equation 3.5:

Finally the entropy weight for every criteria can be calculated by using the results from equations 3.4 and 3.5 and plugging it in the following equation 3.6:

$$f_{ij} = \frac{r_{ij}^{'}}{\sum_{i=1}^{m} r_{ij}}, (i = 1,..., m; j = 1,..., n)$$

Figure 3.5: Entropy Weighted Model 2 [124]

$$w_j = \frac{1 - H_j}{n - \sum_{j=1}^{n} H_j}, \sum_{j=1}^{n} w_j = 1, (j = 1,..., n)$$

Figure 3.6: Entropy Weighted Model 3 [124]

4 Results

In this chapter I have discussed the implementation, tools and technology details. The experimental setup is discussed using the experiment controller and decision cotnroller. Results from decision controller by applying different decision models are presented. Finally the experimental setup including experiment controller and decision controller is analyzed and evaluated.

4.1 Experimental Setup

Experimental setup includes Stackstorm; an event driven automation system based on if-this-then-that principle, OpenStack testbed with deployed Video on demand service, an experiment controller and a decision controller. These components are visualized in the figure 3.2 presented in Contributions chapter 3. Firstly, all the tools used for this experimental setup are described and then there is a brief explanation of each component, its utility and advantages.

4.1.1 Tools and their versions

For this thesis, I have worked with the following tools and technologies. Its important to list the versions for each tool here for reproducing the work

- Stackstorm (st2) 2.10.4 on Python 2.7.6
- Integration packs from Stackstorm Exchange
  - Openstack 0.7.2
  - Ansible 0.5.2
- Ansible 2.7.1
- OpenStack 3.14.2
- Jupyter Notebook 5.4.1
- Docker 18.09.0, build 4d60db4
- Python 2.7
- Python libraries
  - Pandas

- Numpy
- Math
- Requests
- JSON
- Time
- Random
- String
- Scikit-Criteria

4.1.2 Video Streaming Service

Video Service is part of the video-on-demand service implemented by CIT [92]. The following testbed-scenarios [132] are the Ansible deployment scripts for deploying Video On Demand Service. The following figure 4.1 explains a simple model of CIT's video on demand service stack with one client, one load balancer and 3 video servers.

Ideally in a video streaming service environment, the CSP have no control over the client machines. Therefore here for the sake of experiments there is an underlying assumption that RTMP streams are to be generated from clients in case of users streaming this service. The simulated anomaly by generating many RTMP streams in parallel and calculation of quality of service from the client discussed in the upcoming sections.

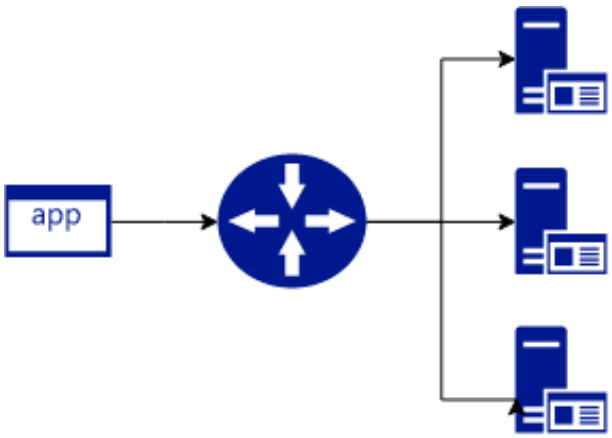

Figure 4.1: Video Service

Load balancer, video servers and clients are docker based services deployed on each of these components as seen in figure 4.1. Following are the components of the video on demand service:

- Video Servers
  Video Servers in our deployment run NGINX-RTMP. NGINX is an open source web

server commonly implemented as reverse proxy, load balancer, HTTP cache and lightweight file server [94] RTMP(Real time messaging protocol) is an streaming protocol for audio, videos and data between server and client over the internet [93].

- Load Balancer
  Load Balancer uses NGINX based haproxy [94]; high availability proxy that provides load balancing capabilities over HTTP and TCP.

- Client
  Client mimics a normal video service user by generating RTMP streams to load the video servers. In our setup, the videos consists of a duration of maximum five minutes.

4.2 Stackstorm Configuration

Stackstorm is available as a Docker container [95]. By default the container contains and configures

- Stackstorm cores services(st2 + st2web + st2mistral)
- mongo
- rabbitmq
- redis

The placement and working of StackStorm in terms of Experiment Controller and OpenStack is visualized using the following figure 4.2

Figure 4.2: Stackstorm as part of Experimental Setup

Ideally all requests should go through Stackstorm therefore a Stackstorm docker container is used here so that the same experiments can be replicated on any other cloud configuration. After installation and configuration and running docker container locally, the GUI for Stackstorm is available at

```
https://localhost
```

In the next subsections I have briefly discussed the implementations and configuration of Stackstorm components for my thesis work.

4.2.1 Integration Packs

The first step of Stackstorm configuration is to install the required integration packs. I have used the following for my experiments:

- Ansible
- Core
- Default
- Openstack

Out of these, Core and Default comes pre-installed while I had to install Ansible and Openstack packs. Packs can be installed using st2 service or from the Stackstorm GUI.

4.2.2 Rules

Rules lie that the heart of StackStorm automation. Rules trigger actions to run either locally or on OpenStack. Rules are defined in YAML syntax which is discussed in Background chapter 2.4.6. HTTP webhooks (part of Stackstorm) [75] can be tied to a rule as trigger. Rules with webhooks as triggers can be set in both GUI and as YAML configuration files. Rules follow the IFTTT principle. An example rule is visualized in terms of the IFTTT principle in the following figure 4.3

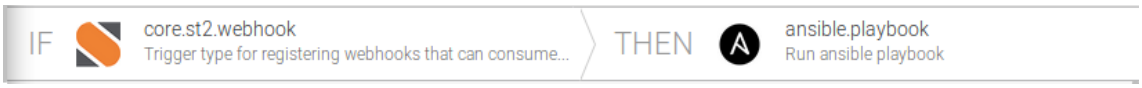

Figure 4.3: If-this-then-that in practice

Rules can be defined in a YAML file. Below is an example start docker service rule defined as YAML in StackStorm. After the code, a brief description of main properties listed in this YAML file is given.

```
{
  "uid": "rule:ansible:ansible_script_create_server_config",
  "tags": [],
  "type": {
    "ref": "standard",
    "parameters": {}
  },
  "enabled": true,
  "trigger": {
    "type": "core.st2.webhook",
    "ref": "core.cb7f8931-25c0-4c47-978f-e19f48520b9d",
    "description": "Trigger type for registering
    webhooks that can consume arbitrary payload.",
```

```
    "parameters": {
      "url": "vidserver"
    }
  },
  "metadata_file": "",
  "context": {
    "user": "st2admin"
  },
  "criteria": {},
  "action": {
    "ref": "ansible.playbook",
    "description": "Run ansible playbook",
    "parameters": {
      "private_key": "/hani.pem",
      "inventory_file": "{{trigger.body.ip}},",
      "verbose": "v",
      "sudo": true,
      "flush_cache": true,
      "playbook": "/playbook.yml",
      "env": {
        "ANSIBLE_HOST_KEY_CHECKING": "False"
      },
      "become": true
    }
  },
  "pack": "ansible",
  "ref": "ansible.ansible_script_create_server_config",
  "id": "5cf9d6bd6417530306ef35a2",
  "name": "ansible_script_create_server_config"
}
```

The following rule is part of the Ansible integration pack. **core.st2.webhook** is used as service which enables this rule to be triggered using a REST call with **vidserver** endpoint. Action to run in this case is the Ansible playbook. **Playbook.yml** to be played contains YAML code to start video service in one of the specified hosts. Inventory file options are passed as payload in the POST request with property **IP**. Finally, set Ansible property **become** to **true** to execute with privileges. All webhooks can be used to trigger rule based actions by using the following REST endpoint:

```
https://localhost/api/v1/webhooks/{URL}
```

The URL parameter added as a suffix to https:../webhooks/ creates the URL to post data to Stackstorm server running as docker container [75]. The request payload is in JSON format.

Pack	Rule	Action
Ansible	Start Video Server via Ansible Script	Run Ansible Playbook
Ansible	Start Client via Ansible Script	Run Ansible Playbook
Core	Reload load balancer configuration	Run Linux command
Core	Get Stackstorm token	Run Linux command to fetch token
Openstack	Associate Floating IP	Openstack server add floating IP
Openstack	Create Instance Snapshot	Openstack snapshot create
Openstack	Create new Instance from Snapshot	Openstack Server Create
Openstack	Instance Hard Reboot	Openstack Server Reboot –hard
Openstack	Instance Soft Reboot	Openstack Server Reboot
Openstack	Service Resize	Openstack Service Resize
Openstack	Service Resize confirm	Openstack Service Resize
Openstack	Stop server	Openstack stop server

Table 4.1: Stackstorm Rules and Actions

Some of the required and optional parameters of defining a rule include name, pack, description, enabled state(true or false), trigger, criteria (this can match payload to a certain criteria/type/pattern and execute actions accordingly), action etc. The rule above can be executed by running the following POST request using curl

```
curl –X POST https://localhost/api/v1/webhooks/vidserver –H

"X–Auth–Token: matoken" –H "Content–Type: application/json"

–d '{"ip": 0.0.0.0}'
```

For the purpose of my experiments I created the following rules 4.1 in Stackstorm server

All the above rules configurations are available in Github [133] and DockerHuB [134] along with other thesis work.

The last piece of experiment controller before running the test scenarios is an anomaly injection. This will be discussed in the next section.

4.3 Anomaly Injection

For simulating anomalies on the VoD service I have used an anomaly injector agent based system developed by CIT department which is build upon several system anomaly simulation tools including Stress-Ng [97]. This stress simulator agent runs as a service in docker container and listens to a a given IP and port for system anomaly simulation requests. You can send REST requests (POST, GET, DELETE) to a running injector

agent. Following are the list of anomalies that can be injected using this injector agent.

- memory leak
- fluctuate memory stress memory
- stress CPU
- leak CPU
- fluctuate CPU
- packet loss
- bandwidth
- packet duplicate
- packet corruption
- packet reordering etc..

Next I have categorized and discussed the simulated anomalies used for my experiments. In the context of this thesis a micro anomaly is defined as an anomaly simulated on one particular component of the video-on-demand service. On the other hand a simulated anomaly effecting the entire state of the system is called macro anomaly. Let's have a look at the anomalies simulated for these experiments and categorize them accordingly:

- Stop VM
  This is a micro anomaly injected directly on specified VM to shut it down. Anomaly injector agent is not used for this simulation, rather a previously discussed procedure via experiment controller to Stackstorm to an Openstack server stop action is run to stop the VM.

- Stress CPU
  This is one of the simplest anomaly that can simulated using the injector agent. By specifying the host, number of cores and percentage of CPU to stress, this anomaly can be simulated. This anomaly can be simulated at both micro and macro level but for this thesis I have used it as a micro anomaly by injecting to a specified video server. Here is an example of CURL to start a stress CPU anomaly with 2 cores and 80 percent CPU resource utilization:

```
curl -X POST --header 'Content-Type: application/json'
--header 'Accept: application/json' -d
'{"parameter":"--cpu=2 --cpu-load=80"}'
0.0.0.0:7999/api/anomalies/stress_cpu/
```

- Generate RTMP streams
  This is a macro anomaly simulated using the client by running a HTTP POST request to the RTMP client service by following this syntax:

  ```
  http://0.0.0.0:7777/api/streams?num={number of RTMP streams}
  ```

  The example shows generating RTMP streams as client node if the client service is running locally and the collector API is running at port 7777 (which is the case in our VoD deployment) and specify the number of streams as a suffix to this URL. In the QoS section 4.4.3 more details of the HTTP load generator are given.

4.4 Experiment Controller

The experiment controller controls the experiments performed in this thesis. It does the major chunk of the practical work including triggering Stackstorm actions via webhooks, logging results, managing data and logs, managing tokens and other related work.

4.4.1 Recovery Workflows

Recovery workflows are aimed at bringing the system to it's normal state or overall improving the state of the system. There can be numerous healing approaches or recovery workflows to fix an anomaly. In the context of this thesis, a recovery workflow is a combination of Stackstorm rules and actions. The following diagram 4.4 explains the working of a recovery workflow

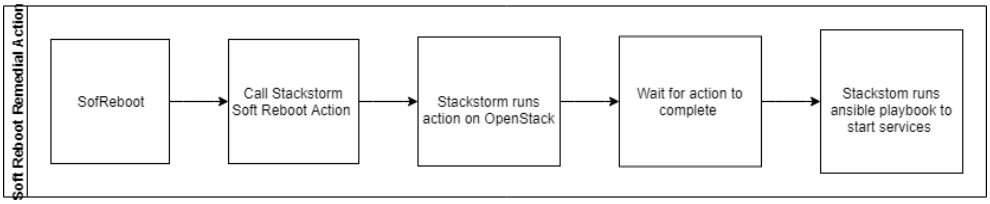

Figure 4.4: Soft Reboot

For my experiments, I implemented the following recovery workflows in the experiment controller.

- Soft Reboot The following diagram 4.4 describes the soft reboot recovery workflow. The process goes as follows
  1. Experiment Controller sends a POST request to Stackstorm server to soft reboot a specified VM
  2. Stackstorm on receiving the requests triggers OpenStack server stop reboot action

3. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
4. Experiment Controller then sends another POST request to Stackstorm which runs Ansible playbook to start video and injector services on the restarted VM

- **Hard Reboot** The process of hard reboot is same as soft reboot but with the exception that a *hard* flag is assigned before executing OpenStack server reboot action

- **Server Resize** The process for Server Resize goes as follows
  1. Experiment Controller sends a POST request with new size as payload to Stackstorm server to resize a given VM
  2. Stackstorm on receiving the request triggers OpenStack server resize action
  3. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
  4. Experiment Controller then sends another POST request to confirm VM resize
  5. Stackstorm on receiving the request triggers OpenStack server resize action
  6. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
  7. Since VM is restarted in this case too, Experiment Controller then sends another POST request to Stackstorm which runs Ansible playbook to start video and injector services on the VM

- **Service Restart**
  1. Experiment Controller sends a POST request to restart service
  2. Stackstorm on receiving the request triggers OpenStack service restart
  3. Experiment Controller waits for the action to complete on OpenStack and Stackstorm

- **Service Scale Up**
  1. Experiment Controller sends a POST request to scale up VoD service
  2. Stackstorm sends a request to create a new VM
  3. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
  4. Experiment Controller then sends another POST request to add the new server IP in load balancer
  5. Stackstorm on receiving the request triggers script to add server in load balancer

6. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
7. Since a new VM is started in this case, Experiment Controller then sends another POST request to Stackstorm which runs Ansible playbook to start video and injector service on the new VM

- **Server Migrate**
  1. Experiment Controller sends a POST request to stop the effected VM
  2. StackStorm forwards server stop request to OpenStack
  3. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
  4. Experiment Controller sends another POST request to create VM snapshot
  5. StackStorm forwards create snapshot request to OpenStack
  6. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
  7. Experiment Controller sends another POST request to create new VM from snapshot
  8. StackStorm forwards create request to OpenStack
  9. Experiment Controller waits for the action to complete on OpenStack and Stackstorm
  10. Since a new VM is started in this case, Experiment Controller then sends another POST request to Stackstorm which runs Ansible playbook to start video and injector services on the migrated VM

4.4.2 Test Scenarios

To simulate a self-recovery approach, tests were conducted using simulating anomalies, recovery workflows and finally logging results. There were three scenarios based on three anomalies discussed in the anomaly section 4.3 and six recovery workflows 4.4.1, therefore in total I have at-least 9 test scenarios to work with. A test scenario based on soft reboot recovery workflow is given in figure 4.5

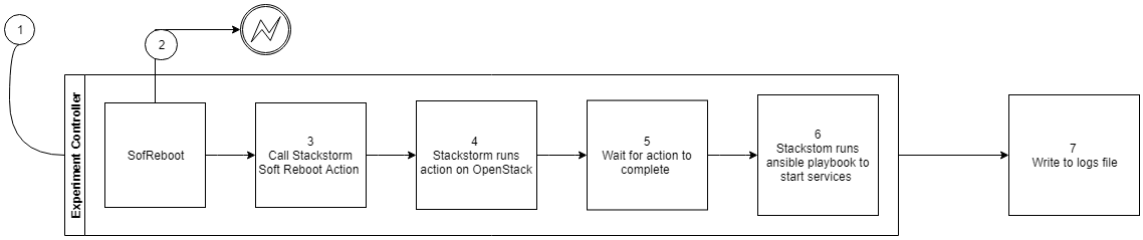

Figure 4.5: A test scenario

The steps in this test scenario are explained as follows

1. Calculate the normal state of the system. In the scope of my thesis work, this is the measurement of QoS
2. Inject the anomaly. This action varies depending on the type of anomaly discussed in the anomaly subsection 4.3.
3. HTTP POST request from Experiment Controller to Stackstorm server. Here's is an example using Python requests library as implemented in Experiment Controller:

```
headers = {'Content-type': 'application/json',
'Accept': 'application/json',
          'X-Auth-Token': 'mytoken'}
payload = {'name':name}
req=r.post('https://localhost/api/v1/webhooks/servers
oftreboot',data=json.dumps(payload),headers=headers,v
erify=False)
```

4. StackStorm server generates request to runs an OpenStack server reboot action on the specified VM
5. Wait for action completion(this is a custom defined property to establish async communication between Stackstorm, Experiment Controller and OpenStack)
6. On the soft reboot action successful completion another request from Experiment Controller to Stackstorm is made to run Ansible playbook using the Ansible integration pack. This playbook starts the services (nginx-rtmp and anomaly-injector-agent) because all docker based services are stopped on VM reboot.
7. All the relevant logs, time-stamps and other calculations are written to an external log file

Execution of test scenarios is logged in terms of different workflow metrics. Other then workflow statistics, impact QoS is also considered to evaluate a workflow. The next subsection describes the implementation of QoS metric calculation.

4.4.3 Calculating Quality of Service

Client VM in our configuration runs a load generator or RTMP streams server for monitoring and provides statistics for quality of service [98]. The client VM in the Experiment Controller is configured to write 1 row/second of log data. This stream statistics client produces file output as a binary file which can be converted to CSV to visualize results. Here is an extract from client.csv file generated by the stream statistics client 4.6

	time tags	streams	openConnections	receivingConnections	opened	closed	errors	bytes	packets	opened/s	closed/s	errors/s	bytes/s	packets/s	packetDelay
2019-07-27 21:29:20.552	NaN	64.0	64.0	64.0	806.0	742.0	0.0	816532036.0	287806.0	3.999461	0.000000	0.0	1.476522e+07	5112.311540	0.012689
2019-07-27 21:29:21.552	NaN	64.0	64.0	64.0	806.0	742.0	0.0	827423211.0	292412.0	0.000000	0.000000	0.0	1.088984e+07	4605.437266	0.013923
2019-07-27 21:29:22.552	NaN	64.0	57.0	57.0	806.0	749.0	0.0	837781025.0	296961.0	0.000000	6.998871	0.0	1.035614e+07	4548.266192	0.013921
2019-07-27 21:29:23.552	NaN	64.0	33.0	31.0	839.0	806.0	0.0	854737753.0	302424.0	32.995202	56.991713	0.0	1.695426e+07	5462.205730	0.006644
2019-07-27 21:29:24.552	NaN	64.0	64.0	64.0	870.0	806.0	0.0	878267082.0	309595.0	30.995973	0.000000	0.0	2.352627e+07	7170.068415	0.007914

Figure 4.6: Extract from the stream statistics client file output

Quality of Service (QoS) is defined as bytes/seconds divided by the current number of open streams. Bytes/seconds gives a QoS value that is an equivalent to frames/seconds which is an important metric for any video streaming service. This value can be averaged over all client nodes in order get an estimation of the QoS of the whole system. Using this metric, when executing some actions to fix anomalies, this value could be observed (using aggregate over a time window). A change in value means that the recovery workflow had an impact on the QoS. The following measurements are made for analyzing QoS using the statistics metrics calculated using stream statistics client:

- Normal State
  Value of QoS expressed as bytes/sec before running a test scenario

- Average impact
  Average impact on the QoS because of a recovery worflow observes the averaged aggregated value of QoS over the execution time frame, right from injecting an anomaly to completion of a recovery workflow.

- Difference in QoS
  Calculates the difference from the time frame after injecting anomaly to the time frame after recovery workflow is complete. 5 minutes waiting time is given before taking the last QoS reading since the videos on VoD service are of maximum length of 5 minutes. Positive values for this metric indicates an improvement in QoS.

- Number of errors
  Number of errors is determined by errors column from the statistics output 4.6. The statistics client log errors on failed connection requests from client to VoD service. This value aggregated over the time frame of a recovery workflow is also an important QoS metric that is considered in this thesis.

In the next section, I have discussed the details of implementation of the decision controller

4.5 Decision Controller

As seen in Figure 1.1 the decision step is a precursor to execution. Decision making involves taking the right decision about the self-recovery action/auto-remediation step to take based on analysis on the input data and some methodology. In the context of this thesis, I experimented with 9 test cases and analyzed the results, applied AI based modelling approaches and selected the best option to execute based on the results. Therefore the decision controller here is used to apply decision models.

4.5.1 Aggregated Results

This sections takes a look into the logs generated by executing test scenarios. To present my results, I have selected Stop VM anomaly and ran it against all six recovery workflows. The aggregated results shown here are parsed and aggregated data values. Selecting the right values to log, parsing log files, cleaning files and aggregating them for modelling purpose was a tedious and time consuming task. For most of the cases, my expected results matched with the actual aggregated results. I have tried to explain and summarize the comparison of expected and actual results using these figures from aggregated results

action_time	6.000000
service_restart_time	0.000000
get_ip_time	0.000003
total_time	25.581866
qos_difference	-45443.327281
qos_average	235093.446125
qos_error_percentage	0.373711
normal_qos	228421.963163

Figure 4.7: Soft Reboot

The following figure 4.7 presents the aggregated results of running soft reboot recovery workflow on a simulated stop VM anomaly. Soft reboot OpenStack API action triggered by Stackstorm fails because soft reboot is not possible on a stopped VM. This results in an expected outcome of negative impact on QoS observed and Services (video anomaly injector) fails to restart.

action_time	17.000000
service_restart_time	31.000000
get_ip_time	33.141676
total_time	84.728058
qos_difference	16934.010517
qos_average	237047.206678
qos_error_percentage	0.001219
normal_qos	226538.779028

Figure 4.8: Hard Reboot

Results from execution of hard reboot recovery workflow is presented in following figure 4.8. Hard reboot OpenStack API action triggered by Stackstorm ends successfully. There is a positive impact on QoS observed and all actions within the hard reboot recovery workflow completes successfully.

action_time	18.000000
action_confirm_time	6.000000
total_time	47.179675
ansbie_run	0.000000
qos_difference	-53051.536285
qos_average	236976.184171
qos_error_percentage	1.945856
normal_qos	230238.380912

Figure 4.9: Resize

The following figure 4.9 presents the aggregated results of running VM resize recovery workflow on a simulated stop VM anomaly. Reize OpenStack API action triggered by Stackstorm ends as a failure because even after resize the VM is not started. This results in an expected outcome of negative impact on QoS observed and Services (video injector) fails to restart.

total_time	2.664301
qos_difference	-62294.450846
qos_average	255066.324790
qos_error_percentage	1.619087

Figure 4.10: Restart Service

Restart Service workflow aggregated results are shown in the following figure 4.10. Restart service action triggered by Stackstorm ends as a failure because restarting service is not possible on a stopped VM. This results in an expected outcome of negative impact on QoS.

The following figure 4.11 presents the aggregated results of running scale up recovery workflow on a simulated stop VM anomaly. Scale Up recovery workflow using multiple Stackstorm to OpenStack API calls ends as successful. There is a positive impact on QoS observed and all actions within the scale up recovery workflow completes successfully.

The following figure 4.12 presents the aggregated results of running migration recovery workflow on a simulated stop VM anomaly. Migrate VM recovery workflow using multiple Stackstorm to OpenStack API calls ends as successful. There is a positive impact on QoS observed and all actions within the migrate recovery workflow completes successfully generating logs.

total_time	154.247029
create_new_snapshot	24.000000
create_new_vm	8.000000
assign_floating_ip	54.000000
ansible_run	34.431835
qos_difference	51609.558954
qos_average	236801.445267
qos_error_percentage	0.001889
normal_qos	251949.852912

Figure 4.11: Scale Up

action	84.853236
total_time	23.000000
create_new_vm	6.000000
assign_floating_ip	6.000000
ansible_run	1.227992
qos_difference	4122.361648
qos_average	237110.188650
qos_error_percentage	0.001177
normal_qos	228519.870643

Figure 4.12: Migrate

In the next subsection lets try to understand the log parameters used to produce these aggregated results

4.5.2 Understanding results

This section tries to develop an understanding based on the results of the above experiments. First let's have a look at the logged values that are common for every recovery workflow.

- **Total time** calculates the total time taken by the test to complete, right from the first Stackstorm action to writing logs
- **Action time** is the time taken by the primary recovery action. For example the primary recovery action in Soft Reboot recovery workflow is Stackstorm rule-based action on Openstack to soft reboot VM. All other actions including service restart are secondary.
- **QoS difference** is the difference in time before and after the recovery workflow. QoS2 (QoS after recovery workflow) - QoS1(after anomaly injection and QoS before recovery workflow) is the difference in QoS. A positive value indicates a drop in QoS, near zero value represent no change while a negative value represent an improvement in QoS.

- **QoS error percentage** is the averaged errors/records over the time frame from running the anomalies to the completion of test scenario.
- **QoS average** is the averaged value of QoS over the time frame of test scenario.

Some of the logged values that are specific to particular recovery workflows are

- **Service restart time or Ansible run** is the time of running Ansible playbook to start video and anomaly injector services
- **Get IP time** logs the time taken to get an available floating IP to assign
- **Action confirm time** confirms the resize after resize action. This is an OpenStack specific behaviour
- **Create new snapshot** for migration
- **Create new VM** to scale up or migrate
- **Assign floating IP** to newly created VM

Now with the results from simulated test scenarios, we can experiment with AI-based approaches to self recovery within the decision controller.

4.5.3 Selecting Parameters

As discussed in chapter 2 in MCDM, including Weighted Sum Model and TOPSIS evaluates alternatives based on criterion. Each criteria is assigned a certain weight based on some parameters. Alternatives to be evaluated are the six recovery workflows discussed in 4.4.1 while a set of five criterion were experimented with to test the validity and working of these models. In the next sections I'll present implementations of Weighted Sum Model,TOPSIS and Entropy weights. For the first two experiments weights are intuitively assigned by experts. Weight assignment and auto-adjustment is further discussed in Evaluation 4.6.

4.5.4 Implementing methodologies for decision system

The implementation of MCDM models; TOPSIS and simple weighted sum model uses similar parameters such as selecting criterion, alternatives and assigning weights. Using the similarities between TOPSIS and Weighted Sum Model, a decision matrix is created. This decision matrix is supplied with criteria, alternative and weight vectors. The selected criterion are

- Mean Time to Repair
- Number of Steps in a auto-remediation workflow
- Impact on quality of service

- Error Percentage
- Success Probability

The significance of using these as criterion is discussed in Evaluation 4.6 section. Next I intuitively assign weights to the criterion as follows

- MTTR weight = 0.1
- Number of Steps weight = 0.1
- Impact weight = 0.1
- Error Percentage weight = 0.1
- Success Probability weight = 0.5

The decision matrix formed by the three vectors using experimental data is given in the following figure 4.13

ALT./CRIT.	MTTR (min) W.0.1	Number of Steps (min) W.0.1	Impact (max) W.0.1	Error Percentage (min) W.0.1	Success Probability (max) W.0.5
soft reboot	25.5819	3	16851.1	0.373711	0
hard reboot	84.7281	3	79228.5	0.0012185	100
resize	47.1797	4	9242.91	1.94586	0
migrate	23	6	66416.8	0.00117718	100
scaleUp	154.247	5	113904	0.00188861	100
restart_service	2.6643	1	0	1.61909	0

Figure 4.13: Decision Matrix

After creating the decision matrix with alternatives, weights and criteria, the next thing to apply different models

4.5.5 Weighted Sum Model

The results of applying the weighted sum model on the decision matrix results in the following 4.14

Since the weighted sum model by default only takes the maximization criteria (highest number is the winner) the computed model first inverts all minimum criterion to maximum. According to the computed weighted sum model server migrate is going to be the recovery workflow which should recover the VoD service with minimum overhead.

4.5.6 TOPSIS

As discussed in 3.3.2 the TOPSIS model can be computed using two normalization criterion namely vector and sum. Applying the TOPSIS model using vector normalization on data gives out the following results 4.15

ALT./CRIT.	MTTR (min) W.0.1	Number of Steps (min) W.0.1	Impact (max) W.0.1	Error Percentage (min) W.0.1	Success Probability (max) W.0.5	Rank
soft reboot	25.5819	3	16851.1	0.373711	0	5
hard reboot	84.7281	3	79228.5	0.0012185	100	2
resize	47.1797	4	9242.91	1.94586	0	6
migrate	23	6	66416.8	0.00117718	100	1
scaleUp	154.247	5	113904	0.00188861	100	3
restart_service	2.6643	1	0	1.61909	0	4

Figure 4.14: Weighted Sum Model

TOPSIS (mnorm=vector, wnorm=sum) - Solution:

ALT./CRIT.	MTTR (min) W.0.1	Number of Steps (min) W.0.1	Impact (max) W.0.1	Error Percentage (min) W.0.1	Success Probability (max) W.0.5	Rank
soft reboot	25.5819	3	16851.1	0.373711	0	4
hard reboot	84.7281	3	79228.5	0.0012185	100	1
resize	47.1797	4	9242.91	1.94586	0	6
migrate	23	6	66416.8	0.00117718	100	2
scaleUp	154.247	5	113904	0.00188861	100	3
restart_service	2.6643	1	0	1.61909	0	5

Figure 4.15: TOPSIS - vector normalization

Applying TOPSIS model using sum as normalization criteria gives out the following results 4.16

TOPSIS (mnorm=sum, wnorm=sum) - Solution:

ALT./CRIT.	MTTR (min) W.0.1	Number of Steps (min) W.0.1	Impact (max) W.0.1	Error Percentage (min) W.0.1	Success Probability (max) W.0.5	Rank
soft reboot	25.5819	3	16851.1	0.373711	0	4
hard reboot	84.7281	3	79228.5	0.0012185	100	2
resize	47.1797	4	9242.91	1.94586	0	6
migrate	23	6	66416.8	0.00117718	100	1
scaleUp	154.247	5	113904	0.00188861	100	3
restart_service	2.6643	1	0	1.61909	0	5

Figure 4.16: TOPSIS - sum normalization

Its observed from the results that hard reboot recovery workflow is selected by TOPSIS with vector normalization. TOPSIS with sum normalization selects server migrate recovery workflow. There is more information related to the TOPSIS model other then normalization method, which is ideal solution, worst solution and closeness to ideal solution. These results are given in the following figure 4.17. Ideal and anti-ideal are the best and worst alternatives chosen by TOPSIS and closeness is the degree of how far an alternative is from the worst solution and close to the best solution.

```
Extra(ideal, anti_ideal, closeness)
Ideal: [1.59639702e-03 1.13402303e-02 8.16399697e-02 5.11166144e-05
 3.20750150e-01]
Anti-Ideal: [0.09242179 0.06804138 0.         0.08449511 0.        ]
Closeness: [0.24868212 0.85118055 0.16627517 0.83655618 0.77097522 0.24197464]
```

Figure 4.17: TOPSIS - results

4.5.7 Plotting

To get better insights from data supplied to models and results from the models, I have visualized them by using different plots. The following figure 4.18 looks at the supplied data using a box plot

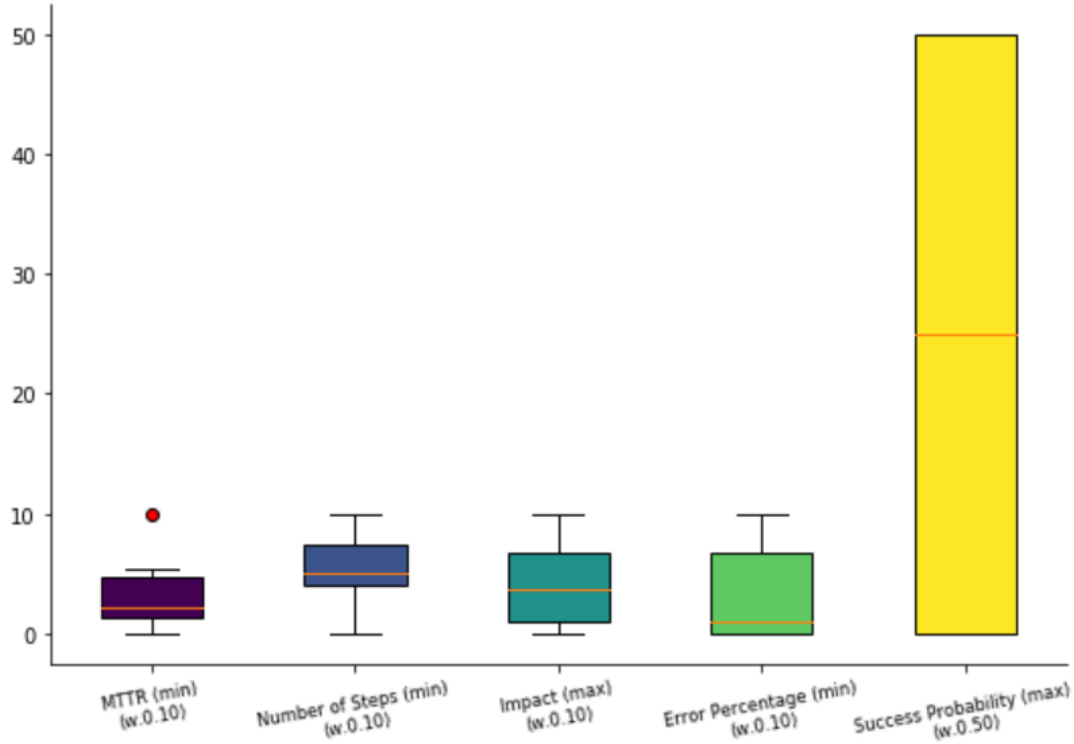

Figure 4.18: Visualizing data

The changes in result of vector and sum normalization in TOPSIS model can also be compared by plotting 4.19:

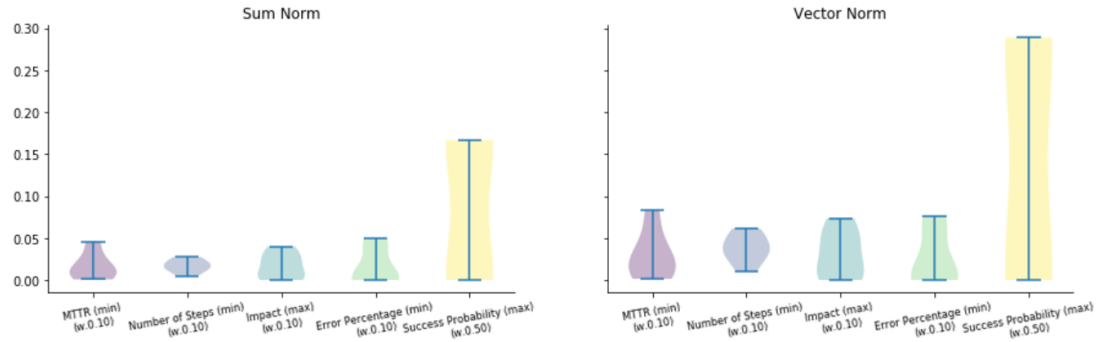

Figure 4.19: TOPSIS - comparison

4.5.8 Entropy Weights and TOPSIS

Entropy Weights Model as discussed previously in 3.3.3 can be used to find weights using scientific methods based on only the data contained in decision matrix. I have implemented TOPSIS model using weights calculated by entropy weights method. The following figure 4.20 shows the application of TOPSIS with entropy weights and vector normalization method. The next figure 4.21 shows the application of TOPSIS with entropy weights and sum normalization method.

TOPSIS (mnorm=vector, wnorm=sum) - Solution:

ALT./CRIT.	MTTR (min) W.0.15842071611451003	Number of Steps (min) W.0.0428533918308015	Impact (max) W.0.18140909915991071	Error Percentage (min) W.0.33928352970355186	Success Probability (max) W.0.27803326319122607	Rank
soft reboot	-0.19557	-0.271695	-0.166971	-0.22332	0	3
hard reboot	-0.347004	-0.271695	-0.3557	-0.00249763	-0.366204	5
resize	-0.275094	-0.309954	-0.111017	-0.348525	0	1
migrate	-0.183085	-0.35435	-0.339194	-0.00242322	-0.366204	6
scaleUp	-0.357829	-0.336728	-0.366618	-0.00366128	-0.366204	4
restart_service	-0.0382298	-0.140502	0	-0.365486	0	2

Figure 4.20: TOPSIS with Entropy weight and vector normalization

TOPSIS (mnorm=sum, wnorm=sum) - Solution:

ALT./CRIT.	MTTR (min) W.0.15842071611451003	Number of Steps (min) W.0.0428533918308015	Impact (max) W.0.18140909915991071	Error Percentage (min) W.0.33928352970355186	Success Probability (max) W.0.27803326319122607	Rank
soft reboot	-0.19557	-0.271695	-0.166971	-0.22332	0	4
hard reboot	-0.347004	-0.271695	-0.3557	-0.00249763	-0.366204	2
resize	-0.275094	-0.309954	-0.111017	-0.348525	0	6
migrate	-0.183085	-0.35435	-0.339194	-0.00242322	-0.366204	1
scaleUp	-0.357829	-0.336728	-0.366618	-0.00366128	-0.366204	3
restart_service	-0.0382298	-0.140502	0	-0.365486	0	5

Figure 4.21: TOPSIS with Entropy weight and sum normalization

In the next section I have evaluated and discussed my design, implementation, exper-

Weighted Sum Model	TOPSIS	Entropy Weights
Intuitive	Similar to WSM	Not intuitive
Simplest	Simple	Comparatively difficult
Built-in in AI-tools	Python implementation available	Work from scratch
Human-assigned weights	Human-assigned weights	Auto-assigned weights
Expert-bias	expert-bias	No expert-bias
Data should be single unit	Multiple class support	Multiple class support
No support for min/max criteria	Support min/max	Support min/max

Table 4.2: Comparison

iments and approaches of self-recovery applied using different decision models.

4.6 Evaluation

A summarized comparison based on my experiments,research on related work(5) and observations, compare the implemented AI-based models in the following table 4.2

There are numerous strategies to decide on model selection and result evaluation in the artificial intelligence space [125]. Despite of the absence of true results(more in Discussion 4.6.1) we can qualitatively evaluate and discuss the AI-based models. According to [125] some of the evaluation questions are

- How accurate the model is
  The accuracy of the models was evaluated by the impact on QoS. Before running a recovery workflow QoS was calculated. This served to benchmark a range of normal/stable QoS. I compared the QoS after recovery workflow selected by the decision model with normal-QoS range. The ranking is based on the difference of observed QoS after workflow execution to the range of normal-QoS.

  1. Rank 1 TOPSIS with vector normalization and human assigned weights selected hard reboot recovery workflow which has the closed value of QoS difference before and after an anomaly
  2. Rank 2 Weighted Sum Model with human assigned weights, TOPSIS sum normalization with human assigned weights and TOPSIS with Entropy weight and sum normalization selected migrate recovery workflow
  3. Rank 3 TOPSIS with entropy weights and vector normalization selected resize

  The underlying assumptions here are that the changes in value of QoS represents positive or negative impact. Frames/second is considered an acceptable criteria to measure QoS and that the normal range of QoS is measured by rigorous system monitoring, experts or defined in SLA.

- How much pre-processing the model needs
  TOPSIS and Weighted Sum Model are linear value functions based on a summation

of scores ,therefore, according to my evaluation, after implementing both TOPSIS and Weighted Sum Model, its eminent that these models require very less pre-processing. After a decision maker decides on the alternatives and criterion, the only data pre-processing part is a simple normalization step.

- How fast the model is
  The speed and efficiency of the models I experimented with replies on the mean time to repair of the recovery workflow (reffered as total time in test scenarios) selected by the model. This is due to the reason that the difference in time for pre-processing, building and execution of these models is negligible.

4.6.1 Discussion

The criterion for evaluation of recovery workflows was selected after careful consideration. According to my understanding and backed by research on the related work(5), mean time to repair and impact on quality of service are important criterion for a cloud based environment. Mean time to repair should be small since in a cloud computing environment we rely on real-time applications and a prolong error situation can result in lost of hundreds of dollars. Quality of Service as discussed in 2.2.2 is an integral part of any Cloud SLA. Therefore the recovery workflows should have less impact on the QoS while execution and depending on the anomaly, an improved quality of service should be observable after running recovery workflows. Connection errors from client to video service are recorded as error percentage. This can indicate a network or bandwidth issue, incorrect load-balancer configuration etc. Another criteria is Success Probability; which records the probability of success for remedial actions within workflows. Finally the number of steps criteria is based on number of actions inside a recovery workflow. This is important because my designed solution relies heavily on communication between experiment controller, Stackstorm and OpenStack and if less number of connection requests are desired in a cloud-environment, the recovery workflows with smaller number of intermediate steps should be selected.

Regarding weights, I evaluated TOPSIS and weighted sum model by implementing them with the assumption of an expert assigned weight vector. These methods relied on initial human input, knowledge and intuitive sense of decision experts. There is no method inherently available for automated adjustment for weights in weighted sum model and TOPSIS. The assumption here is that the weight vector is a result of expert knowledge on criterion and weights, based on a set of auto-remediation pairs and that the overall decision of a lot of experts is optimal. But as discussed in [124] the assumption of using expert weights or human assigned weights is prone to certain problems.

Therefore, considering the implementation of closed loop automation in my thesis with a focus on less human intervention by using self-recovery approaches, there had to be a way of determining weights automatically. For this reason Entropy weight method was used to mathematically assign weight vector to TOPSIS. This objective fixed weight

method determines weight on information obtained from criterion eliminating human disturbances to make results more accurate [124] However, for the scope of this thesis weight calculation and adjustment remains a qualitative consideration since there is no data to produce quantitative results.

In the context of my thesis, with no expert data, in my opinion there is no single answer when it comes to comparing results of different MCDM methods (TOPSIS and Weighted Sum Model in our case). This is because that in our testbed scenario the true result is unknown. If the true results were known, the use of MCDM could not have been justified.

4.6.2 Limitations

There are some limitations to the implementation and evaluation of the AI-based models in this thesis

- True data
  There are no validated data-sets available to evaluate the impact of self-recovery approaches on this video-on-demand service. Absence of data limits the number of approaches that can be evaluated on the topic of self-recovery in cloud systems. It also makes difficult to quantify selected remediation workflows as correct or incorrect.

- Heterogeneity
  The implemented and evaluated models do not entirely cover the heterogeneous aspects of a cloud system. However the proposed solution can be replicated to solutions other then OpenStack.

5 State of the art

AIOps is an emerging field with lots of research going on around and number of CSPs are involved to offer competitive solutions. Folks at Microsoft highlighted the challenges and innovation at AIOps will likely bring [111]. They proposed an AIOps solution based on Microsoft Azure covering all four aspects of AIOps namely monitoring, analyzing, decision and execution [111]. Baidu [112]. outlined their approach to AIOps by uncovering a solution which Incorporated anomaly detection, traffic scheduling, root cause analysis, trend forecasting and other data-mining and machine learning algorithms under one hood. A recent book by the name AIOps: Predictive Analytics & Machine Learning in Operations [113] discusses on the topic of AIOps and how it differs from IT operations and how it is going to bring innovation in the IT market.

An extensive review on fault-tolerance methods [114] in cloud systems was written by Mukwevho, Mukosi Abraham and Celik Turgay. According to the survey, there has been only a limited amount of research in determining the cloud's ability to learn and adapt from faults. It considers the challenges of providing fault-tolerance in cloud systems such as heterogeneity and a lack of standard, service downtimes, allowed data to be lost and important data to always recover, cloud native and other workloads. The survey categorizes fault detection and recovery methods into three; reactive, proactive and resilient. It states that the commonly applied reactive method mostly involves restarting. Resilient methods are described as ones predicting faults, implement methods to avoid faults, monitor and tuning cloud systems by interacting with it intelligently. Here [114] self-healing technique is defined as a feature of the cloud system which enables it to detect, diagnose and repair faults. Dobson et al. [115] envisioned the necessity of autonomic computing for cloud systems with a self-healing approach in place for system engineering [116], emphasized on self-recovery property for service-oriented applications to aid automatic recovery techniques. They [116] developed a knowledge based approach using service agents that provide information on the state of the service. Knowledge on the decision of self-recovery comprises of this real-time agent information and user defined quality of service parameters.

Dai, Yuanshun and Xiang, Yanping and Zhan Gewei [102] were the first to realize the importance of self-healing and self-diagnosis in cloud systems. They [102] proposed a hybrid tool and showed the effectiveness using multivariate decision making and Naive Bayes classifier. Bala, Anju and Chana Inderveer [103] studied fault-tolerance in cloud systems with emphasis on virtualization to enable load-balancing. A data-replication technique is used here [103]. Nallur, Vivek and Bahsoon, Rami and Yao Xin proposed a self-optimization architecture driven approach to maintain the promised quality of ser-

vice based on utility theory and service level agreements [104].

A cloud self-healing framework was proposed by Alhosban, Amal and Hashmi, Khayyam and Malik, Zaki and Medjahed, Brahim [105] that dynamically evaluated the performance of running services on a cloud and developed recovery plans accordingly. This [105] was a preventive approach which selected the execution plan and executed it right then and there or saved it for later execution. Another similar framework but focused on only web based application deployed on cloud was propesed by Magalhães, João Paulo and Silva Luís Moura [106]. This framework detects cloud workload and performance anomalies from the user's perspective and try to inform the CSP so that changes or improvements can be made accordingly.

Azaiez, Meriem and Chainbi, Walid [107] proposed a multi-agent system for intelligent and autonomous cloud. The proposed system arguably works without external intervention and is able to interact with the Cloud infrastructure to analyze it's state. Based on the analysis either a migration, check-pointing or replicating strategy is adopted to recover from the problem of failed resources. Mosallanejad et al. [110] proposed a model to develop heirarchical SLAs for the purpose of self-healing in cloud systems. Input the SLA requirements and the proposed model will send alerts on a violation. Xin [109] in his PhD thesis suggested a preventive approach by combining data analysis and machine learning techniques to predict failure scenarios in cloud.

RADAR, an autonomic cloud resource management technique, proposed by Gill, Sukhpal Singh and Chana, Inderveer and Singh, Maninder and Buyya Rajkumar [108] is one of the closed related work to my thesis. This comprehensive study aims at delivering cost-efficient and reliable cost services. RADAR focuses on self-healing by both diagnosing the flaw and taking remedial action. The outcome is evaluated in terms of important SLA properties including but not limited to execution cost, resource contention and time. RADAR [108] monitors quality of service regularly based on the defined parameters during the execution of a workload (any network or data change to the service) and then either generates alerts or perform appropriate actions to preserve the system's state.

Number of solutions and tools, both enterprise and open-source are available in the DevOps and AIOps domain promising self-recovery and auto-remediation for better quality of service. Netflix developed Winston [117]; an event-driven automation platform based on Stackstorm. Winston enables automatic diagnostic and remediation in response to faulty events. OpenStack offers a self-healing model for its infrastructure management using Zaqar message queue and mistral workflow service [118]. CloudCheckr[119] offers cloud automation at scale to manage auto-scaling and self-healing by monitoring cloud systems for costs and efficiency. Kubernetes [120]; a tool to manage clusters of containers, pushes the use of containers since they inherently offer self-healing capabilities. Its easier to manage the health and configuration of a container and in most of the cases it can restart without effecting the entire state of the system. Kubernetes [120] offers

self-healing capabilities based on these properties of containers. Cloud conformity [121] auto-re-mediates security risks for AWS infrastructure.

I have tried to cover the available solutions and tools to the best of my knowledge. It is important to note here that this AIOps area is wide [122] and evolving therefore a complete survey of all available tools and techniques is not possible. Model and learning structures are continuously updated to cater for changing dynamics of cloud systems.

6 Conclusion

This thesis presented the design and implementation of a recovery engine with decision making approaches and connected it with an event-driven automation system. A video on demand service running on OpenStack served as the testbed. Stackstorm was used as the event driven automation system tool which is based on the if-this-then-that principle. Stackstom ran as a service on a docker container locally for experiments. Recovery workflows and actions were created on Stackstorm and executed on the video on demand service running on OpenStack. These workflows and actions were tied to a webhook and could be triggered with a REST call.

In the next phase an experimental setup was designed and implemented which simulated test scenarios by injecting anomalies and executing recovery workflows. Data was collected by running three different anomalies on six different recovery workflows. Workflows were evaluated based on metrics such as mean time to repair, success probability and impact on quality of service.

In the last phase different AI-based decision making approaches were implemented based on the data from experiments. Decision making approaches; Weighted Sum Model, TOPSIS and entropy based weights with TOPSIS were implemented. For weighted sum model and TOPSIS a weight vector based on the assumption of available expert knowledge was used. In the later part, an entropy based weight calculation was applied to remove human bias. Recovery workflows were evaluated using multiple criterion based on the applied weights. In the end, results from different decision making approaches were evaluated.

List of Acronyms

API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CIT	Complex and Distributed IT Systems
CSP	Cloud Service Provider
EC2	Amazon's Elastic Compute Cloud
EDA	Event-Driven Architecture
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IaC	Infrastructure as Code
IaaS	Infrastructure as a Service
IFTTT	if-this-then-that
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
MCDA	Multi-criteria decision analysis
MCDM	Multi-criteria decision making
MTTR	Mean time to Repair
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
QoS	Quality of Service
REST	Representational State Transfer
SaaS	Software as a Service
SCM	Software Configuration Management
SDLC	Software Development Life Cycle
SDK	Software Developer Kit
SLA	Service Level Agreement
TCP	Transmission Control Protocol
URL	Universal Resource Location
VM	Virtual Machine
VoD	Video on Demand
YAML	Yet another markup language
XML	Extensible Markup Language

Bibliography

[1] P. M. Mell and T. Grance, "The nist definition of cloud computing," no. 1, 2011.

[2] R. G. A. D. J. R. K. A. K. G. L. D. P. A. R. I. S. Michael Armbrust, Armando Fox and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," 2009.

[3] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1.

[4] "Build the future of open infrastructure.." `https://www.openstack.org/`. Accessed: 2019-08-10.

[5] "User stories showing how the world." `https://www.openstack.org/user-stories/`. Accessed: 2019-08-11.

[6] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: Toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 6, p. 38–42, 2012.

[7] T. Rosado and J. Bernardino, "An overview of openstack architecture," *Proceedings of the 18th International Database Engineering Applications Symposium on - IDEAS 14*, no. 7, 2014.

[8] "Conceptual architecture." `https://docs.openstack.org/install-guide/get-started-conceptual-architecture.html`. Accessed: 2019-08-09.

[9] "Openstack keystone service." `https://docs.openstack.org/keystone/latest/`. Accessed: 2019-08-01.

[10] "Openstack identity service." `https://docs.openstack.org/api-ref/identity/index.html`. Accessed: 2019-08-02.

[11] "Openstack nova." `https://docs.openstack.org/nova/latest/`. Accessed: 2019-08-04.

[12] "It operations (information technology operations)." `https://cio-wiki.org/wiki/IT_Operations_(Information_Technology_Operations)`. Accessed: 2019-08-06.

[13] "Evolve your it operations team for cloud success." `https://searchcloudcomputing.techtarget.com/tip/Evolving-IT-operations-teams-for-cloud-success`. Accessed: 2019-08-06.

[14] "Disciplined agile - disciplined agile it (dait)." https://disciplinedagileconsortium.org/Disciplined-Agile-IT-DAIT. Accessed: 2019-08-06.

[15] "It operations." http://disciplinedagiledelivery.com/agility-at-scale/it-operations/. Accessed: 2019-08-08.

[16] F. Dynamics, "The impact of automation on it operations," Tech. Rep. 15, Freeform Dynamics,Fujitsu, 2017.

[17] "What is a site reliability engineer and why you should consider this career path." https://opensource.com/article/18/10/what-site-reliability-engineer. Accessed: 2019-08-14.

[18] "Site reliability engineering." SiteReliabilityEngineering. Accessed: 2019-08-14.

[19] "The evolution of the site reliability engineer." https://thenewstack.io/the-evolution-of-the-site-reliability-engineer-sre/, note = Accessed: 2019-08-14.

[20] J. Waller, N. C. Ehmke, and W. Hasselbring, "Including performance benchmarks into continuous integration to enable devops," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 2, p. 1–4, 2015.

[21] B. S. Farroha and D. L. Farroha, "A framework for managing mission needs, compliance, and trust in the devops environment," in *Proceedings of the 2014 IEEE Military Communications Conference*, MILCOM '14, (Washington, DC, USA), pp. 288–293, IEEE Computer Society, 2014.

[22] A. Dyck, R. Penners, and H. Lichter, "Towards definitions for release engineering and devops," 05 2015.

[23] J. Humble and J. Molesky, "Why enterprises must adopt devops to enable continuous delivery," *The Journal of Information Technology Management*, p. 3–6, Aug 2011.

[24] "The evolution of the site reliability engineerwhat is devops? "in simple english"." https://blog.usejournal.com/what-is-devops-in-simple-english-6550fbb129bd. Accessed: 2019-07-23.

[25] "Devops." https://www.happiestminds.com/whitepapers/devops.pdf. Accessed: 2019-07-21.

[26] "What is cloud sla (cloud service-level agreement)? - definition from whatis.com." https://searchstorage.techtarget.com/definition/cloud-storage-SLA. Accessed: 2019-07-22.

[27] "kurz sla?." https://www.cloudcomputing-insider.de/was-ist-ein-service-level-agreement-kurz-sla-a-574461/. Accessed: 2019-07-22.

[28] "Cloud management: Zur rolle von slas." https://www.computerwoche.de/a/zur-rolle-von-slas. Accessed: 2019-07-22.

[29] "5 things to look for in a cloud service level agreement." https://solutionsreview.com/cloud-platforms/5-things-to-look-for-in-a-cloud-service-level-agreement/. Accessed: 2019-07-22.

[30] "What is automated testing?." https://smartbear.com/learn/automated-testing/what-is-automated-testing/. Accessed: 2019-07-25.

[31] "Automation testing tutorial: What is, process, benefits tools." https://www.guru99.com/automation-testing.html. Accessed: 2019-07-25.

[32] "Best automation testing tools for 2019 (top 10 reviews)." https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2. Accessed: 2019-07-25.

[33] "Was ist continuous integration?." https://www.dev-insider.de/was-ist-continuous-integration-a-690914/. Accessed: 2019-07-26.

[34] "Continuous integration." https://martinfowler.com/articles/continuousIntegration.html. Accessed: 2019-07-26.

[35] "What is continuous integration?: Continuous integration using jenkins." https://www.edureka.co/blog/continuous-integration/. Accessed: 2019-07-26.

[36] "Continuous integration." https://www.ionos.de/digitalguide/websites/web-entwicklung/continuous-integration/. Accessed: 2019-07-27.

[37] "Was versteht man unter ci/cd?." https://www.redhat.com/de/topics/devops/what-is-ci-cd. Accessed: 2019-07-27.

[38] "Was ist continuous deployment?." https://www.dev-insider.de/was-ist-continuous-deployment-a-652804/. Accessed: 2019-07-27.

[39] "I want to do continuous deployment." https://devops.com/i-want-to-do-continuous-deployment/. Accessed: 2019-07-27.

[40] "I want to do continuous deployment21 automated deployment tools you should know - dzone devops." https://dzone.com/articles/21-automated-deployment-tools-you-should-know. Accessed: 2019-07-27.

[41] "Was ist continuous delivery?." https://www.dev-insider.de/was-ist-continuous-delivery-a-664429/. Accessed: 2019-07-27.

[42] "Eine einführung in continuous delivery, teil 1: Grundlagen." https://www.heise.de/developer/artikel/Eine-Einfuehrung-in-Continuous-Delivery-Teil-1-Grundlagen-2176380.html?seite=all. Accessed: 2019-07-28.

[43] "Continuous delivery." https://continuousdelivery.com/, note = Accessed: 2019-07-28.

[44] "Business software and services reviews."

[45] "Was ist scm?." https://www.dev-insider.de/was-ist-scm-a-623224/. Accessed: 2019-07-29.

[46] standard.

[47] "Software configuration management: Patterns, best practices, and tools for agile and devops." https://stackify.com/software-configuration-management-patterns/. Accessed: 2019-07-29.

[48] "Software configuration management: Patterns, best practices, and tools for agile and devops10 best software configuration management tools (scm tools in 2019)."

[49] M. J. Scheepers, "Virtualization and containerization of application infrastructure: A comparison," in *21st Twente Student Conference on IT*, vol. 1, pp. 1–7, 2014.

[50] P. P. Kukade and G. Kale, "Auto-scaling of micro-services using containerization," *International Journal of Science and Research (IJSR)*, pp. 1960–1964, 2013.

[51] O. Pozdniakova and D. Mazeika, "A cloud software isolation and cross-platform portability methods," pp. 1–6, 04 2017.

[52] A. Mouat, *Using Docker: Developing and Deploying Software with Containers.* " O'Reilly Media, Inc.", 2015.

[53] "What is a container?." https://www.docker.com/resources/what-container. Accessed: 2019-07-30.

[54] "What is cloud orchestration (cloud orchestrator)? - definition from whatis.com." https://searchitoperations.techtarget.com/definition/cloud-orchestrator. Accessed: 2019-07-30.

[55] "The best tools for cloud infrastructure automation." https://blog.newrelic.com/engineering/best-cloud-infrastructure-automation-tools/. Accessed: 2019-07-31.

[56] Di Liu and Pin Wang, "Use python api to automate script based on open stack platform," in *2015 12th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pp. 465–468, Dec 2015.

[57] "Was ist infrastructure as code (iac)? - definition von whatis.com." https://www.computerweekly.com/de/definition/Infrastructure-as-Code-IAC. Accessed: 2019-07-31.

[58] "Was ist infrastructure as code? – hpe glossar." https://www.hpe.com/de/de/what-is/infrastructure-as-code.html. Accessed: 2019-07-31.

[59] "Was ist iac?." https://www.dev-insider.de/was-ist-iac-a-727301/. Accessed: 2019-07-31.

[60] "Ansible is simple it automation." https://www.ansible.com. Accessed: 2019-08-01.

[61] "How ansible works." https://www.ansible.com/overview/how-ansible-works. Accessed: 2019-08-02.

[62] "Yet another markup language." https://yaml.org/spec/history/2001-03-30.html. Accessed: 2019-08-02.

[63] "What do you mean by "event-driven"?." https://martinfowler.com/articles/201701-event-driven.html. Accessed: 2019-08-02.

[64] "What is event-driven architecture (eda)? - definition from whatis.com." https://searchmicroservices.techtarget.com/definition/event-driven-architecture-EDA. Accessed: 2019-08-02.

[65] D. group, "White Paper: Event-Driven Architecture," tech. rep., Daitan, 2017.

[66] "Articulating the business value of event-driven architecture." https://www.gartner.com/en/documents/3723455/articulating-the-business-value-of-event-driven-architec. Accessed: 2019-08-02.

[67] "Event-driven architecture (eda)." https://www.gartner.com/it-glossary/eda-event-driven-architecture/. Accessed: 2019-08-02.

[68] "The importance of event-driven automation - dzone devops." https://dzone.com/articles/the-importance-of-event-driven-automation. Accessed: 2019-08-02.

[69] "How to use wordpress ifttt recipes to automate your workflow." https://kinsta.com/blog/wordpress-ifttt-recipes/. Accessed: 2019-08-02.

[70] "Ansible v.s. salt (saltstack) v.s. stackstorm." https://medium.com/@anthonypjshaw/ansible-v-s-salt-saltstack-v-s-stackstorm-3d8f57149368. Accessed: 2019-08-02.

[71] "Stackstorm." https://stackstorm.com/. Accessed: 2019-08-02.

[72] "Stackstorm overview." https://docs.stackstorm.com/overview.html. Accessed: 2019-08-02.

[73] "Stackstorm installation overview." https://docs.stackstorm.com/install/overview.html. Accessed: 2019-08-02.

[74] "Stackstorm datastore." https://docs.stackstorm.com/datastore.html, note = Accessed: 2019-08-02.

[75] "Stackstorm webhooks." https://docs.stackstorm.com/webhooks.html. Accessed: 2019-08-02.

[76] "Stackstorm features." https://stackstorm.com/features/. Accessed: 2019-08-02.

[77] "Stackstorm exchange." https://exchange.stackstorm.org/. Accessed: 2019-08-02.

[78] "Stackstorm exchange." https://searchitoperations.techtarget.com/definition/AIOps. Accessed: 2019-08-03.

[79] "An introduction to aiops." https://medium.com/@iauro/an-introduction-to-aiops-f3e063eaf1eb. Accessed: 2019-08-03.

[80] "Market guide for aiops platforms 2019," gartner analyst report, Gartner, Geneva, CH, 2019.

[81] "he top use cases for aiops in enterprise it operations." https://www.information-age.com/aiops-enterprise-use-cases-123481767/. Accessed: 2019-08-04.

[82] "Closed loop automation." https://www.blueplanet.com/resources/what-is-closed-loop-automation.html. Accessed: 2019-08-04.

[83] "What is closed loop control system? - definition from whatis.com." https://whatis.techtarget.com/definition/closed-loop-control-system. Accessed: 2019-08-05.

[84] "Control systems." http://www.ent.mrt.ac.lk/~rohan/teaching/EN5001/Reading/DORFCH1.pdf. Accessed: 2019-08-05.

[85] "A primer on closed-loop automation." https://www.networkworld.com/article/3294204/a-primer-on-closed-loop-automation.html. Accessed: 2019-08-05.

[86] "A primer on closed-loop automationchoosing the right closed loop implementation strategy." https://www.teoco.com/latest/news/blog/closed-loop-strategy/. Accessed: 2019-08-06.

[87] M. Pavan and R. Todeschini, "Multicriteria decision-making methods," *Comprehensive Chemometrics*, p. 591–629, 2009.

[88] "An overview of multi-criteria analysis techniques," *Multi-criteria analysis: a manual*, p. 18–19, Jan 2009.

[89] "Case-based reasoning: Was ist das eigentlich? [techniken der künstlichen intelligenz]." https://www.empolis.com/blog/kuenstliche-intelligenz/case-based-reasoning/. Accessed: 2019-08-07.

[90] M. Velasquez and P. T. Hester, "An analysis of multi-criteria decision making methods," *International Journal of Operations Research*, vol. 10, no. 2, pp. 56–66, 2013.

[91] X.-S. Qin, G. H. Huang, A. Chakma, X. Nie, and Q. Lin, "A mcdm-based expert system for climate-change impact assessment and adaptation planning–a case study for the georgia basin, canada," *Expert Systems with Applications*, vol. 34, no. 3, pp. 2164–2179, 2008.

[92] "Welcome to the cit chair." https://www.cit.tu-berlin.de/menue/complex_and_distributed_it_systems/. Accessed: 2019-08-08.

[93] "citlab/testbed-scenariosrtmp (real time messaging protocol)." https://www.itwissen.info/RTMP-real-time-messaging-protocol.html. Accessed: 2019-08-15.

[94] "Load balancing with haproxy, nginx and keepalived in linux." https://linuxhandbook.com/load-balancing-setup/. Accessed: 2019-08-15.

[95] "Stackstorm/st2-docker." https://github.com/StackStorm/st2-docker. Accessed: 2019-08-15.

[96] "Stackstorm rules." https://docs.stackstorm.com/rules.html. Accessed: 2019-08-16.

[97] "Ubuntu manpage: stress-ng - a tool to load and stress a computer system." https://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html. Accessed: 2019-08-16.

[98] "antongulenko/stream-statistics-client." https://github.com/antongulenko/stream-statistics-client. Accessed: 2019-08-16.

[99] F. Helff, L. Gruenwald, and L. d'Orazio, "Weighted sum model for multi-objective query optimization for mobile-cloud database environments.," in *EDBT/ICDT Workshops*, 2016.

[100] E. Triantaphyllou, *Multi-criteria decision making methods: a comparative study*. Springer, 2011.

[101] A. Assari and E. Assari, "Role of public participation in sustainability of historical city: usage of topsis method," *Indian Journal of Science and Technology*, vol. 5, no. 3, pp. 2289–2294, 2012.

[102] Y. Dai, Y. Xiang, and G. Zhang, "Self-healing and hybrid diagnosis in cloud computing," in *IEEE International Conference on Cloud Computing*, pp. 45–56, Springer, 2009.

[103] A. Bala and I. Chana, "Fault tolerance-challenges, techniques and implementation in cloud computing," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 1, p. 288, 2012.

[104] V. Nallur, R. Bahsoon, and X. Yao, "Self-optimizing architecture for ensuring quality attributes in the cloud," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, pp. 281–284, IEEE, 2009.

[105] A. Alhosban, K. Hashmi, Z. Malik, and B. Medjahed, "Self-healing framework for cloud-based services," in *2013 ACS International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–7, IEEE, 2013.

[106] J. P. Magalhães and L. M. Silva, "A framework for self-healing and self-adaptation of cloud-hosted web-based applications," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1, pp. 555–564, IEEE, 2013.

[107] M. Azaiez and W. Chainbi, "A multi-agent system architecture for self-healing cloud infrastructure," in *Proceedings of the International Conference on Internet of things and Cloud Computing*, p. 7, ACM, 2016.

[108] S. S. Gill, I. Chana, M. Singh, and R. Buyya, "Radar: Self-configuring and self-healing in resource management for enhancing quality of cloud services," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 1, p. e4834, 2019.

[109] R. Xin, "Self-healing cloud applications," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 389–390, IEEE, 2016.

[110] A. Mosallanejad, R. Atan, M. A. Murad, and R. Abdullah, "A hierarchical self-healing sla for cloud computing," *International Journal of Digital Information and Wireless Communications (IJDIWC)*, vol. 4, no. 1, pp. 43–52, 2014.

[111] Y. Dang, Q. Lin, and P. Huang, "Aiops: real-world challenges and research innovations," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, pp. 4–5, IEEE Press, 2019.

[112] X. Qu and J. Ha, "Next generation of devops: Aiops in practice@ baidu," 2017.

[113] A. Masood and A. Hashmi, "Aiops: Predictive analytics & machine learning in operations," in *Cognitive Computing Recipes*, pp. 359–382, Springer, 2019.

[114] M. A. Mukwevho and T. Celik, "Toward a smart cloud: A review of fault-tolerance methods in cloud systems," *IEEE Transactions on Services Computing*, 2018.

[115] S. Dobson, R. Sterritt, P. Nixon, and M. Hinchey, "Fulfilling the vision of autonomic computing," *Computer*, vol. 43, no. 1, pp. 35–41, 2010.

[116] R. Angarita, M. Rukoz, M. Manouvrier, and Y. Cardinale, "A knowledge-based approach for self-healing service-oriented applications," in *Proceedings of the 8th International Conference on Management of Digital EcoSystems*, pp. 1–8, ACM, 2016.

[117] "Introducing winston - event driven diagnostic and remediation platform." `https://medium.com/netflix-techblog/introducing-winston-event-driven-diagnostic-and-remediation-platform-46ce39aa81cc`. Accessed: 2019-08-16.

[118] "Self-healing sig." `https://wiki.openstack.org/wiki/Self-healing_SIG`. Accessed: 2019-08-16.

[119] "Cloud automation, security, compliance, cost optimization." `https://cloudcheckr.com/cloud-automation/`. Accessed: 2019-08-16.

[120] "stratoscale." `https://www.stratoscale.com/blog/kubernetes/auto-healing-containers-kubernetes/`. Accessed: 2019-08-16.

[121] "Auto-remediation for aws." `https://www.cloudconformity.com/solutions/aws/auto-remediation.html`. Accessed: 2019-08-16.

[122] "The ultimate list of ai ops." `https://xebialabs.com/the-ultimate-devops-tool-chest/the-ultimate-list-of-ai-ops/`. Accessed: 2019-08-16.

[123] S. Vajapeyam, "Understanding shannon's entropy metric for information," *arXiv preprint arXiv:1405.2061*, 2014.

[124] X. Li, K. Wang, L. Liu, J. Xin, H. Yang, and C. Gao, "Application of the entropy weight and topsis method in safety evaluation of coal mines," *Procedia Engineering*, vol. 26, pp. 2085–2091, 2011.

[125] "Choosing the right machine learning algorithm." `https://hackernoon.com/choosing-the-right-machine-learning-algorithm-68126944ce1f`. Accessed: 2019-08-16.

[126] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, no. 1, pp. 41–50, 2003.

[127] A. Iosup, X. Zhu, A. Merchant, E. Kalyvianaki, M. Maggio, S. Spinner, T. Abdelzaher, O. Mengshoel, and S. Bouchenak, "Self-awareness of cloud applications," 11 2016.

[128] G. Da Cunha Rodrigues, R. N. Calheiros, V. T. Guimaraes, G. L. d. Santos, M. B. de Carvalho, L. Z. Granville, L. M. R. Tarouco, and R. Buyya, "Monitoring of cloud computing environments: Concepts, solutions, trends, and future directions," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, (New York, NY, USA), pp. 378–383, ACM, 2016.

[129] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.

[130]

[131] G.-H. Tzeng and J.-J. Huang, *Multiple attribute decision making: methods and applications*. Chapman and Hall/CRC, 2011.

[132] "Testbed scenarios." `https://github.com/citlab/testbed-scenarios`. Accessed: 2019-08-20.

[133] "Master thesis github repo." `https://github.com/hanizaidi110/masterthesis`. Accessed: 2019-08-20.

[134] "Stackstorm container configured with openstack." `https://hub.docker.com/r/hanizaidi110/masterthesis`. Accessed: 2019-08-20.