Double Degree

Politecnico di Torino and EURECOM

Master’s Degree in Computer Engineering with specialization in Cloud Computing and Networks @ Poltecnico di Torino - A.a. 2022/2023
Master's Degree in Cybersecurity @ EURECOM – Promo 2025
Graduating session of March 2025

Self-healing Mechanisms

Analysis and Implementation for Stringent-SLAs Products

Supervisors:
REBAUDENGO Maurizio
BALZAROTTI Davide
OBEID Elie

Candidate:
FALCI Emanuele

Abstract English

This thesis explores self-healing mechanisms in cloud applications, with a focus on reducing manual intervention in system reliability engineering (SRE). Conducted as part of an internship at Amadeus, the study examines existing self-healing solutions, evaluates their integration within the company's technology stack, and proposes improvements to automated incident remediation. The research covers built-in self-healing features of key infrastructure components such as Couchbase, Oracle DB, Kubernetes, and cloud platforms. Additionally, it investigates external monitoring and automation tools, including ARGOS, ServiceNow, and AWX, to enhance alert management and remediation processes. A key outcome of this work is the enhancement of Amadeus' "Auto Remediation of Alerts" framework, improving its ability to autonomously resolve common incidents. The findings contribute to the broader field of SRE by showing how automation can improve system availability, reduce operational costs, and enhance response times to failures.

Abstract French

Ce mémoire explore les mécanismes d'auto-réparation dans les applications cloud, avec un accent sur la réduction de l'intervention manuelle en ingénierie de fiabilité des systèmes (SRE). Réalisée dans le cadre d'un stage chez Amadeus, cette étude examine les solutions d'auto-réparation existantes, évalue leur intégration dans l'environnement technologique de l'entreprise et propose des améliorations aux processus de remédiation automatique des incidents. La recherche couvre les fonctionnalités d'auto-réparation intégrées à des composants clés tels que Couchbase, Oracle DB, Kubernetes et les plateformes cloud. De plus, elle examine les outils externes de surveillance et d'automatisation, notamment ARGOS, ServiceNow et AWX, pour améliorer la gestion des alertes et les processus de remédiation. Un résultat clé de ce travail est l'amélioration du cadre "Auto Remediation of Alerts" d'Amadeus, améliorant sa capacité à résoudre automatiquement les incidents courants. Les conclusions de cette étude contribuent au domaine du SRE en montrant comment l'automatisation peut améliorer la disponibilité des systèmes, réduire les coûts opérationnels et améliorer les temps de réponse aux pannes.

Contents

I Introduction **1**

1 Host Organization **2**
1.1 AMADEUS . . . 2
1.2 Hospitality Sector . . . 3

2 Context **4**
2.1 HA, UHA and SLAs . . . 4
2.2 SRE team . . . 6

3 Self-Healing: definition of the problem **8**
3.1 Outline . . . 8

4 Technical background **10**
4.1 Monitoring tools . . . 10
4.1.1 ARGOS . . . 10
4.1.2 Splunk . . . 12
4.2 Incident Management tools . . . 12
4.2.1 Service Now . . . 12
4.2.2 Win@proach . . . 15
4.3 Automation tools . . . 16
4.3.1 AWX . . . 16
4.3.2 Jenkins . . . 16
4.4 Tools in AMADEUS . . . 17
4.4.1 Recovery Orchestrator and Switch Automation . . . 17
4.4.2 Auto Remediation of Alerts . . . 17
4.5 Conclusion . . . 17

II Self-Healing in cloud applications **19**

5 Built-in mechanisms **20**
5.1 Couchbase . . . 20
5.2 Kubernetes . . . 22
5.3 Azure Kubernetes Service . . . 24
5.4 Conclusion . . . 25

III Self-Healing through monitoring 26

6 Introduction 27

7 "Auto Remediation of Alerts" - simple use cases 29
7.1 Monitoring: ARGOS Escalation . . . 29
7.2 Automation: AWX template . . . 31
7.3 Orchestration: SNow . . . 32
7.3.1 SNow Configuration Table . . . 33
7.3.2 SNow Workflow . . . 33
7.4 Limits and improvements . . . 35
7.4.1 Improvement: ansible limit . . . 35
7.5 Couchbase use case . . . 36
7.5.1 ARGOS alert . . . 36
7.5.2 AWX template . . . 37
7.5.3 SNow configuration table . . . 38
7.5.4 Testing . . . 39
7.5.5 Troubleshooting . . . 39
7.5.6 Pushing to PRD . . . 40

8 SRE Remediation workflow - complex use cases 41
8.1 Design . . . 41
8.2 Implementation details . . . 43
8.2.1 Integration with AWX . . . 44
8.2.2 integration with Jenkins . . . 44
8.2.3 integration with MSTeams . . . 46
8.2.4 Splunk escalation . . . 47

IV Conclusion 48

9 Conclusion and future works 49

Part I

Introduction

Chapter 1

Host Organization

This chapter provides an understanding of the current business of AMADEUS. Particular focus will be placed on the hospitality section (HOS).

1.1 AMADEUS

AMADEUS, established in 1987 through a collaboration between Air France, Lufthansa, Iberia, and the Scandinavian Airline System, has grown into a global leader in the travel and tourism industry. With its headquarters in Madrid, Spain, with a major development site in Sophia Antipolis, France, AMADEUS employs more than 19,000 people around the world and serves customers in more than 190 countries through 173 local AMADEUS Commercial Organizations (ACO) [2]. The company's mission is to shape the future of travel by creating innovative technology solutions that enhance travelers' experience and streamline operations for industry stakeholders.

AMADEUS offers a comprehensive suite of services tailored to various sectors within the travel industry.

For travel agencies, AMADEUS provides booking engines, travel management software, and customer relationship management systems, enabling agencies to manage their operations efficiently. Airlines benefit from AMADEUS passenger service systems, revenue management tools, and departure control systems, which optimize their operations and improve passenger experiences. In addition, AMADEUS offers airport management systems, passenger processing systems, and baggage handling solutions to streamline airport operations.

In the hospitality sector, AMADEUS develops business intelligence, central reservation systems, and guest management tools, helping hotels manage their properties and improve guest satisfaction. The company also provides booking and management solutions for railway and ground transportation providers, ensuring seamless travel experiences across different modes of transport.

A key aspect of AMADEUS' success is its commitment to innovation and sustainability. The company invests heavily in research and development to stay ahead of industry trends and meet the evolving needs of its clients. AMADEUS fosters a collaborative and innovative work environment, emphasizing diversity, sustainability, and continuous learning. This approach has allowed AMADEUS to drive digital transformation and connectivity within the travel industry, positioning itself as a leading provider of technology solutions.

The extensive reach of AMADEUS and the breadth of its services are evident in its impressive portfolio of clients. The company currently serves more than 400 airlines, 184 airport operators, over 1 million hotel properties, 240 tour operators, 79 ground handlers, and more than 25 rail operators. These numbers show AMADEUS' ability to cater to a diverse clientele and its commitment to adapting to the evolving needs of the travel industry. As AMADEUS continues to expand its client base and innovate, it remains dedicated to driving digital transformation and connectivity, ultimately shaping the future of travel[1].

1.2 Hospitality Sector

AMADEUS Hospitality Platform (AHP) hosts a wide range of products and services designed to improve guest experience and streamline hotel operations. The central tool developed for several hotels chains, such as IHG and MGM is aCRS(AMADEUS Central Reservation System). This system is the core piece of software that processes shopping and booking requests from the various chains hotels. In simple terms, it is behind the different websites when planning your trip. In 2025, two new customer chains, Marriott and Accor, will be launched.

Also mid-market hotel chains can benefit from a set of smaller scale products, or use iHotelier CRS which is meant to for medium-size or independent customers.

Just to give a glance at the technical part, this system runs on a microservices architecture, and contains multiple bricks hosted on Azure. It is meant to provide a high availability service to customers and currently manage 1MTPS. The datastore layer is based mainly on Couchbase and Oracle, for longer-term storage, it relies on Flexify/S3. The computing part runs on OpenShift and K8s. Lastly, the network depends on a mix of components such as Equinix, F5, Venafi, and others.

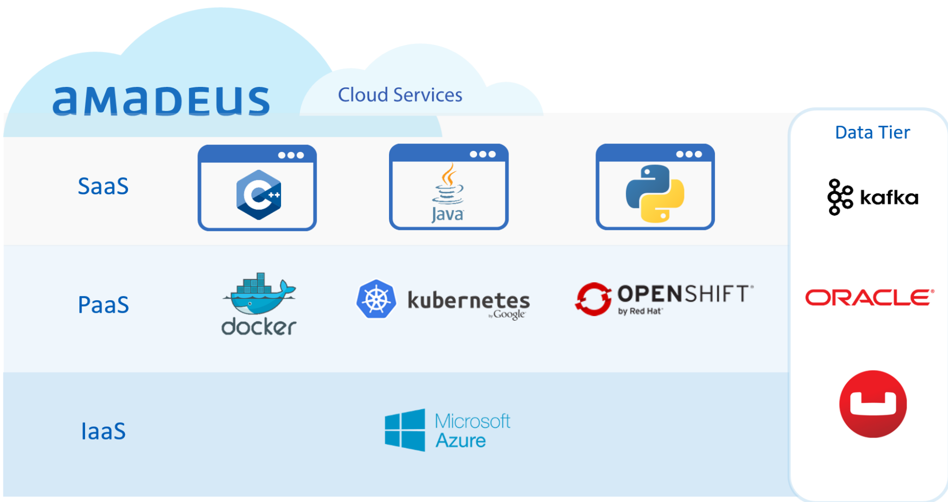


Figure 1.1: High-Level Design of aCRS[11]

Chapter 2

Context

The work was carried out during an internship in the Hospitality Site Reliability Engineering (SRE) team. In this chapter, the concepts of High Availability, Ultra High Availability, and Service Level Agreements (SLAs) will be introduced. Then also the role of SRE team will be explained.

Even though these concepts are not strictly technical, they will provide a better understanding of the significance of the SRE job for the company, and the financial impact of failures. Therefore, it highlights the importance of having a way to solve the issues as soon as possible and the reason to develop a self-healing mechanism.

2.1 HA, UHA and SLAs

This section will introduce the concepts of High Availability (HA), Ultra High Availability (UHA), and Service Level Agreements (SLAs).

Most of the products in AMADEUS are designed to provide a high level of availability to customers. HA and UHA are crucial design principles in technology and business contexts aimed at ensuring the reliability and performance of services.

HA refers to systems designed to be operational and accessible for a significant percentage of time. In some AMADEUS products it goes up to 99.95% of the time[10]. This typically involves incorporating redundancy and failover mechanisms to minimize downtime. HA is generally sufficient for applications that are still critical, but where some level of downtime could be acceptable.

On the other hand, UHA is designed for scenarios where even minimal downtime is unacceptable. It is a more stringent design principle that focuses on minimizing downtime and ensuring continuous service availability. In AMADEUS products the uptime requirement is 99.99% (or higher)[10]. Achieving UHA often requires specific architectural constraints. Examples could be independence from certain technologies that may introduce latency or support for multi-region/multi-platform architectures with active-active or active-standby patterns.

In Hospitality each product has its own specification, for example in figure 2.1 it is possible to see that shopping and booking products require UHA since they are more critical operations, while other products like the administration services generally fall into the HA category.

In order to provide better reliability, the product is replicated in different regions. Taking as an example the aCRS product, it is replicated in West US 2 (wus2) and East US 2 (eus2). Depending on the availability requirements of the specific services, cluster replicas

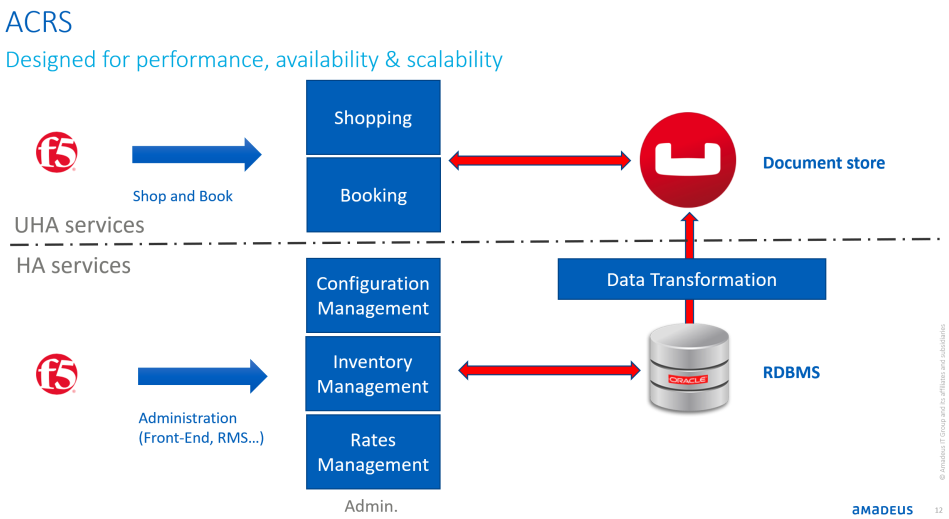


Figure 2.1: Example of HA and UHA services

are configured in different ways. These configurations are active/active, active/standby, or active/passive. Each one is tailored to the criticality and resource needs of the service (respectively, in blue, light blue, and gray in figure 2.2).
For shopping services, an active/active configuration is employed, ensuring that multiple instances are running and handling requests simultaneously in both regions.
In the case of booking services, an active/standby configuration is used. This means that while one instance is actively handling requests, a standby instance is kept up and running ready to take over in case the active instance fails.
For administration services, an active/passive configuration is implemented. Here, the active instance handles all requests, while the passive instance is down and can be turned on if needed.

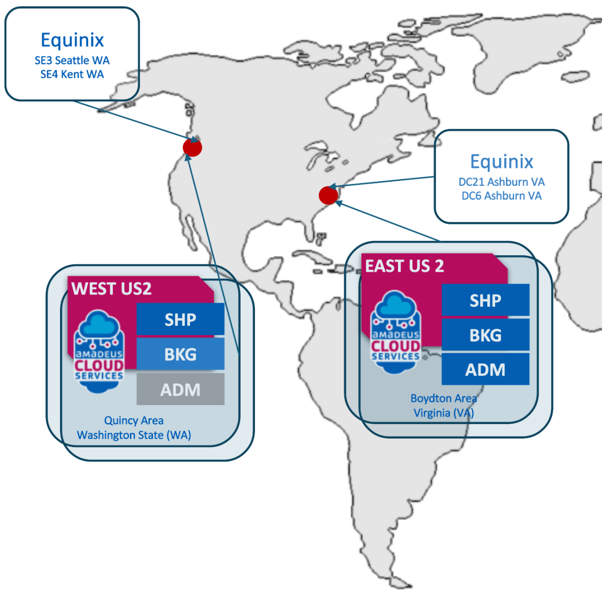


Figure 2.2: aCRS Regions

One other important requirement is independence from some technologies that are not suitable for UHA constraints. For example, in the data storage domain, as can also be seen in figure 2.1, the UHA components should rely only on CouchBase for the storage of information. Oracle cannot be used due to some issues encountered in active/active configuration. Next, there is the independence from any other non-UHA components. If a flow needs to involve a non-UHA component, this must be an asynchronous flow.

The required performance is specified in the Service Level Agreements (SLAs). The SLA is a formal contract between a service provider and a customer that defines the expected level of service. It sets out the metrics by which the service is measured and specifies the remedies or penalties if the agreed-upon service levels are not met. Typically, SLAs include targets for various performance indicators, such as service availability, response times to address issues, recovery times from incidents, and the number and duration of outages. These agreements are essential for setting clear expectations, formalizing reporting procedures, and managing the performance of both parties involved. They also define eventual fees in case the requirements are not met.

In aCRS, SLAs are particularly crucial to ensure the reliability and availability of the hotel platform, thereby enhancing guest satisfaction and operational efficiency. To comply with this principle, some Service Level Objectives (SLO) are defined. For example:
Recovery Time Objective (RTO): defines the maximum acceptable downtime for an application or service in the context of disaster recovery and business continuity. Within 4 hours in certain contexts. It ensures services are restored promptly, minimizing downtime and maintaining operational continuity.

Recovery Point Objective (RPO): indicates the maximum acceptable amount of data loss measured over time. It determines how frequently data backups should occur to ensure data can be recovered within the specified timeframe in the event of a disaster. For this objective, the timings are much shorter, down to minutes.

There are no explicit performance thresholds in the literature that can define strictly what is Ultra-HA and what is just HA. In aCRS, based on the experience of the SLAs of the requested customers, three levels of performance were defined, High availability, medium UHA, or high UHA[17]. In order to assess the compliance of a component or service with the UHA constraints, the rating of some properties like response time and availability, resiliency and recoverability, scalability, technology, and deployment were defined. An example is shown in the hexagonal graph in Figure 2.3, where the component is classified as non-UHA.

2.2 SRE team

I worked on the Hospitality Site Reliability Engineering (SRE) team. SRE is a discipline that incorporates aspects of software engineering and applies them to infrastructure and operations problems. The SRE role was firstly defined at Google in 2003. It was created by Ben Treynor Sloss, who was tasked with making Google's large-scale services more reliable and efficient. He defined:

> ”SRE is what happens when you ask a software engineer to design an operations team”[26]

The primary goal of SRE is to create highly reliable and scalable software systems. They focus on the long-term reliability and scalability of the software in operation,

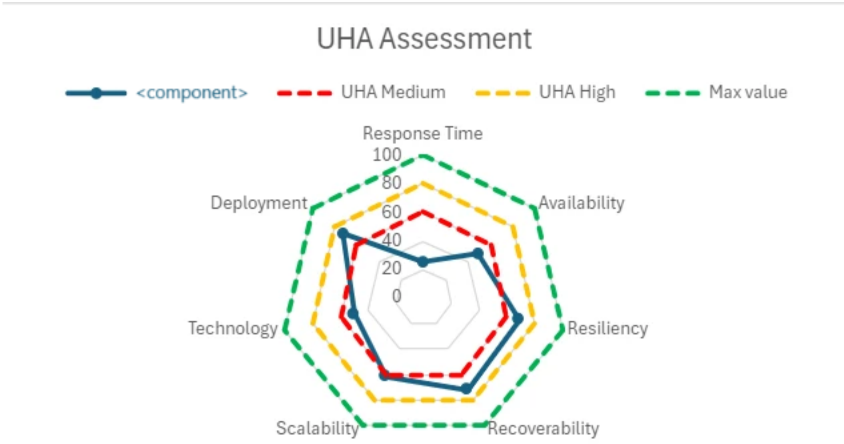


Figure 2.3: UHA Assesment for a component

ensuring that the software applications remain reliable amidst frequent updates from development teams. SRE is often compared to DevOps, but while DevOps focuses on the speed and efficiency of the development process, SRE emphasizes the reliability and scalability of software in operation.

In particular, HOS SRE team's scopes are two. Firstly they have a responsibility over projects during the build phase. They need to deliver the automation and monitoring framework and architecture principles that will address the Ultra High Availability SLAs.

The second scope is an operational responsibility during the run phase. In fact, the unit will be responsible for the continuous delivery of the service to our customer with the agreed SLAs[16].

Given the complexity of the application, many failures can appear, whether it is a planned maintenance activity on the Azure side, on the AMADEUS side, or an (unplanned) incident. The application is designed to survive such failures, with a resilience and redundancy model, but sometimes it requires human intervention.

This thesis scope works more in this second objective, in particular, the purpose is avoiding these manual actions. Therefore, it will provide a comprehensive analysis of self-healing mechanisms.

Chapter 3
Self-Healing: definition of the problem

The main goal of the thesis is to avoid manual intervention in case of failures through self-healing systems. The term self-healing does not have a unique definition. In the literature, it has been used in different contexts. Here are some definitions that will give a holistic view of what self-healing is and what it entails.

Starting with Microsoft, it defines self-healing as: ”the ability of your workload to automatically resolve issues by recovering affected components and, if needed, failing over to redundant infrastructure”[24].

RedHat describes it as an advanced stage of automation that leverages historical data-driven insights and automation to identify and resolve issues in hybrid cloud environments[27].

Lastly, Amazon Web Services defines it as ”an automated system that detects and fixes bugs”[3].

Combining the points of view, it can be stated that self-healing is the ability of the system to automatically detect issues. Then recognize them, possibly leveraging historical insights. Then automatically recover the affected components, if the worst case, by switching to a backup infrastructure.

The implementation of self-healing mechanisms offers numerous advantages.
Reduces the effort of SRE team members, who can spend less time recovering from issues and allocate more resources to other tasks. Moreover, it reduces the mean time to recover from an incident, helping to fulfill the SLAs. On the company side, it also avoids the triggering of on-calls for incidents, reducing the expenses.

Although the benefits of implementing self-healing mechanisms are significant, it is crucial to recognize the potential risks associated with delegating sensitive tasks to automated systems. If not executed correctly, that could potentially cause even more issues. Therefore, careful planning, rigorous testing, and continuous monitoring are essential to ensure the reliability and effectiveness of self-healing mechanisms.

3.1 Outline

This thesis aims firstly to explore existing self-healing mechanisms and to check if they are already implemented in the company or not. It will include mechanisms that come out of the box with the technologies adopted in the company, but also mechanisms that need external monitoring and automation tools. Therefore, it is going to analyze an auto-remediation tool already utilized and configure an use case. Lastly a new mechanism will be implemented for more complex use cases and for further development.

Part I will provide a good understanding of the context in which the thesis is set.

Specifically, it will cover the technologies being utilized and the requirements that need to be met. The focus will be on the instrument used for monitoring, logging, orchestration, and automation, as well as identifying which out-of-the-box solution are provided.

Part II will examine the existing self-healing mechanisms within the various technologies. Examples include a Kubernetes pod able to restart itself in case of failure, an automatic switch between Azure regions, and Couchbase's automatic failover of a node. Since not all applications have built-in self-healing mechanisms, it is necessary to rely on external tools for system monitoring and automated reactions is necessary.

Not everything is covered by the application built-in mechanisms, from there the need to rely on external tools in order to monitor our system and automate a reaction. Part III will explore how to handle these issues: how to monitor the system and automatically respond to alerts. Firstly, a tool already existing in the company will be analyzed, it will be slightly improved, and a use case will be implemented. Then a separate tool with advanced features specific to the team will be designed.

Chapter 4

Technical background

Given that the project aims to improve self-healing mechanisms in AMADEUS HOS, it is essential to understand the existing environment. This chapter will present various auxiliary technologies employed by the company for monitoring, logging, and automating orchestration, which are fundamental for developing a self-healing system. In addition, the incident management process will be examined. By the conclusion of the chapter, there will be a clear understanding of the requirements to be met including technologies available for use and those that need to be integrated.

4.1 Monitoring tools

This section will provide an overview of the monitoring tools utilized within the company.

4.1.1 ARGOS

ARGOS is a proprietary tool that is becoming the standard monitoring solution for AMADEUS applications and platforms. This comprehensive monitoring solution is based on three open source projects. Prometheus, Grafana and Thanos. It is designed to address a wide range of monitoring use cases within the organization.

Prometheus, an open source system monitoring and alerting toolkit originally developed at SoundCloud, has evolved into a robust and widely adopted solution for monitoring a variety of systems and applications. What makes Prometheus unique with respect to other monitoring tools is its capacity of pulling the metric from an exporter, instead of waiting for the metric to be pushed. It so collects and stores metrics as time series data, meaning that metrics information is stored with the timestamp at which it was recorded, along with optional key-value pairs called labels. This approach allows for powerful and flexible querying capabilities, written in their own query language (PromQL), enabling users to gain deep insights into their systems' performance and behavior.

ARGOS utilizes this pull-based system, periodically scraping metrics from various components where an exporter is installed. The components are identified through a service discovery mechanism. The metrics are then stored in Thanos and eventually presented on a Grafana dashboard.

Thanos extends Prometheus' capabilities by offering clustering and long-term storage solutions. It allows for scalable management of Prometheus instances, enabling high availability and efficient storage of historical data. Grafana instead provides a rich set of

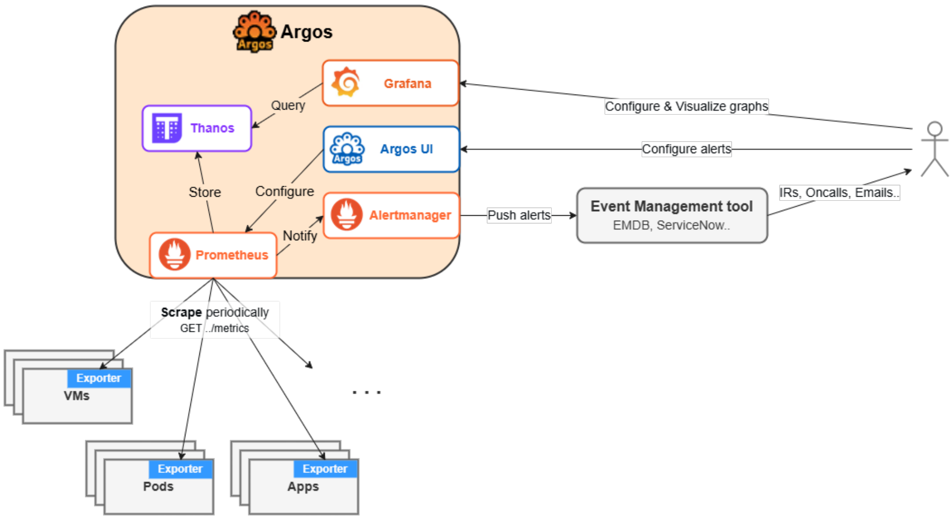


Figure 4.1: ARGOS Architecture Overview

features for visualizing time series data, allowing users to create dynamic and interactive dashboards.

Together, Prometheus, Thanos, and Grafana form a powerful trio for monitoring and observability. Prometheus + Thanos handle the collection, storage, and querying of metrics, while Grafana provides the tools to visualize and analyze this data. This combination enables the organization to monitor its infrastructures and applications, detect anomalies, and respond to incidents promptly, ensuring the reliability and performance of their systems.

ARGPS is well-integrated with other technologies in AMADEUS. Through the ARGOS UI, users can set up alerting rules that escalate to other applications. A common use case is enabling alerts to escalate to different platforms like Win@Proach or ServiceNow, automatically creating incidents under specific conditions or sending emails. Alerts can be defined not only through the ARGOS UI but also as code, using the dashboard-as-code approach. This allows for easy recreation of alerts from the code saved in a repository in case of a major incident in the system[15].

By integrating Thanos with Prometheus and Grafana, the organization can achieve a holistic monitoring solution that supports both real-time and historical data analysis, ensuring robust and scalable observability for their systems.

Dashboard as a code

Dashboard as a Code (DaaC) is an approach that treats the configuration and management of dashboards as code, enabling version control, collaboration, and automation. By defining dashboards in a declarative format, teams can efficiently manage and deploy them across different environments. This method ensures consistency, reduces manual errors, and allows for seamless integration with CI/CD pipelines.

Within AMADEUS, DaaC is implemented using BitBucket to store and manage both dashboard configurations and alerts, thereby facilitating efficient monitoring and alerting processes. The company provides a tool that allows users to easily create a personal project in ARGOS for testing purposes. This tool automatically generates the alert rule from the provided code. Additionally, default functions are available to create basic alerts and escalate them to common tools.

This format will be used in part III.

4.1.2 Splunk

Splunk on the other hand, is an event-based monitoring tool utilized in AMADEUS for log management, analysis and alerting. It is a robust platform designed to handle large volumes of data, offering tools for searching, monitoring, and analyzing machine-generated data in real-time. This platform aids organizations in gaining insights from their data, facilitating issue troubleshooting, ensuring security, and optimizing operations.

Data is collected and indexed from various sources, making it searchable and analyzable. Similar to ARGOS, Splunk employs a specific syntax to perform complex searches and generate reports to identify patterns and trends. Additionally, Splunk allows for the setup of alerts to notify users of specific events or anomalies in the data, which can be escalated to other tools within AMADEUS, such as Win@Proach or ServiceNow.

However, Splunk alerting can be challenging. Incorrectly configured search queries can consume significant resources and slow down the system. In the worst-case scenario, it could result in the loss of some logs due to the system's inability to handle the load. Therefore, Prometheus is the recommended alerting tool whenever possible, as evidenced by the number of alerts directed to ServiceNow (Figure 4.2). Nonetheless, Splunk remains valuable for handling more granular logs of individual components, making it a useful tool in the self-healing mechanism.

4.2 Incident Management tools

To begin, the focus will be on the incident management tools, as they will serve as the primary orchestrators in Part III. The company currently utilizes two instruments for incident management: Win@proach and ServiceNow (SNow). Win@proach, a proprietary solution, is scheduled to be completely replaced by ServiceNow by the end of this year. The following section will provide an overview of ServiceNow, including its functionalities and associated concepts.

4.2.1 Service Now

ServiceNow was founded in 2004 by Fred Luddy, the former CTO of Peregrine Systems and Remedy Corporation. Since its inception, SNow has grown rapidly and has become a leading provider of enterprise cloud solutions. It is a cloud-based workflow automation platform that provides software as a service. It enables enterprise organizations to improve operational efficiencies by streamlining and automating routine work tasks. The company specializes in

- IT services management (ITSM)
- IT operations management (ITOM)
- IT business management (ITBM)
- allowing users to manage projects, teams and customer interactions via a variety of apps, plugins and APIs.

ServiceNow also provides an app store of tool offerings from third parties. Its products can be used to support most workflows because of the wide range of tools the organization offers. Some common ways the products are used include ticketing systems to manage large-scale projects via on-suite ticketing tools, benchmarking to track progress and predictive modeling to manage workflows. IT professionals operating a service desk/help desk can use ServiceNow products to organize their help cases, problem management and instance management. ServiceNow is able to assist with machine learning and artificial intelligence processes that can help for incident management.

AMADEUS, after adopting ServiceNow during its transition to the cloud, it is currently investing on the platform and plans to replace various other tools like win@proach and the Configuration Management DataBase (CMDB). There are distinct instances for different environments - DEV, QA, PREPROD, TRN, PROD - to specializes the workflows and cater to different scopes.

In order to understand how an incident is created and categorized, it is necessary to understand first the following concepts: how the system is monitored, how the configuration of the system is stored, and how to handle the alerts in order to keep only the relevant ones. To comprehend the creation and categorization of an incident, it is essential to first grasp the following concepts: the monitoring of the system, the storage of the system's configuration, and the management of alerts to ensure only relevant ones are retained.

Configuration Management DataBase

Each alert is ideally linked to a Configuration Item in the Configuration Management Database. A Configuration Item (CI) is any Service Component or Infrastructure Element that needs to be managed in order to provide the successful delivery of a service. While the Configuration Management DataBase, is a database used to store information about CIs. SNow provides a CMDB, and most CIs are stored there. Since before adapting SNow, AMADEUS was using the another CMDB provided by another company, the transition is not fully completed yet. For example, the Couchbase cluster's node are not configured as "couchbase node" but simply as k8s nodes. Each CI has a type and attributes that define its characteristics and relationships with other CIs. They also have tags, some of which are mandatory in order to be correctly identified[12]. A CI could be in maintenance mode. In that case, no incident are created.

Given that most resources are hosted in Azure, their configuration has been automated through a discovery mechanism. ServiceNow's discovery process involves a dedicated MID server (concept explained later) that polls discovery tasks from the ECC queue (a ServiceNow-specific queue used for communication between the main instance and the MID server). The MID server requests PaaS details directly from the Kubernetes cluster via the Kubernetes REST API. This MID server operates using a service account with read-only access to the cluster resources[13].

Event Management

Regarding the event management. The event is created because of an event rule that specifies some filters and sets the different fields. Then if not configured otherwise, an alert is created. The alert goes through a correlation rule that can group the alerts together and set a severity. Only the primary alert is then treated. Eventually alert management rules are then applied. For all these rules there is a priority in order to decide which one to apply in case multiple ones match.

A dashboard was created, accessible to whom needs to access information regarding the incident. On the dashboard it is possible to see the different events, alerts and the status of a CI. The role needed is defined in SNow as ITIL (Information Technology Infrastructure Library) role. Most of the event take origin from ARGOS as can be seen from Figure 4.2.

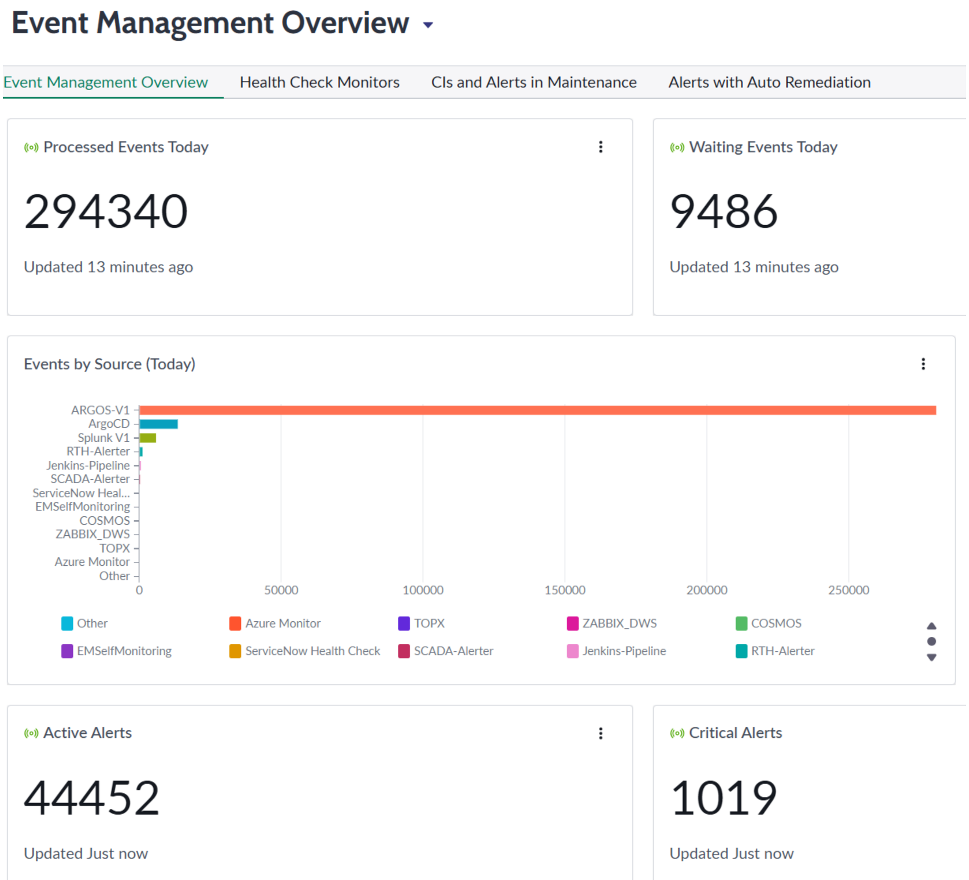
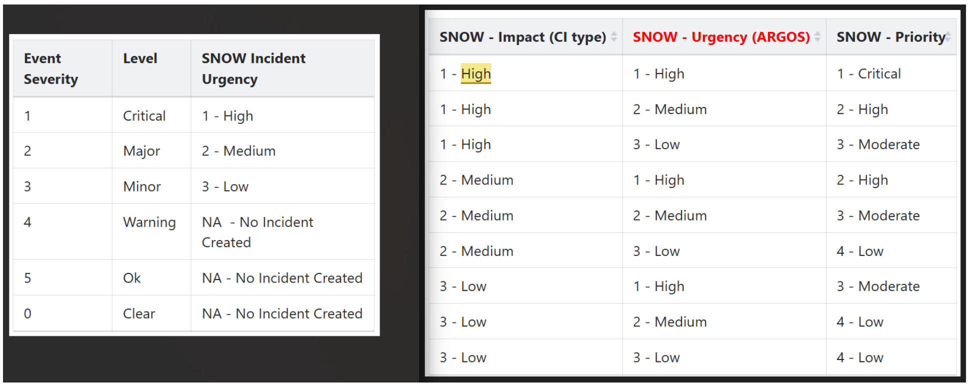


Figure 4.2: ServiceNow Event Dashboard

The severity of an alert in SNow could be critical, high or moderate. In case the alert comes from ARGOS, it is calculated following some tables. Incident Urgency directly corresponds to the Event Severity code sent by Argos[14].

MID server

A Management, Instrumentation, and Discovery (MID) Server is a Java application that operates on a local network server. It facilitates communication and data transfer between a ServiceNow instance and external applications, data sources, and services. MID Servers are used to control and secure ServiceNow's interactions with an organization's systems, particularly those behind a firewall. They support integrations, orchestration, discovery, and script execution. Best practices include deploying MID Servers on Windows for broader compatibility, assigning each MID Server a single use, clustering for load balancing and failover, and placing them close to the resources they communicate with to enhance performance. The MID Server operates as a Windows service or UNIX daemon,



Event Severity	Level	SNOW Incident Urgency
1	Critical	1 - High
2	Major	2 - Medium
3	Minor	3 - Low
4	Warning	NA - No Incident Created
5	Ok	NA - No Incident Created
0	Clear	NA - No Incident Created

SNOW - Impact (CI type)	SNOW - Urgency (ARGOS)	SNOW - Priority
1 - High	1 - High	1 - Critical
1 - High	2 - Medium	2 - High
1 - High	3 - Low	3 - Moderate
2 - Medium	1 - High	2 - High
2 - Medium	2 - Medium	3 - Moderate
2 - Medium	3 - Low	4 - Low
3 - Low	1 - High	3 - Moderate
3 - Low	2 - Medium	4 - Low
3 - Low	3 - Low	4 - Low

Figure 4.3: ServiceNow Event Severity Mapping

initiating communication with the ServiceNow instance via the External Communication Channel (ECC) Queue[28]. Within AMADEUS there are already plenty of MID servers already set up, particularly relevant are the MID servers configured to connect to the global network where there are the instances of AWX and Jenkins.

Service Now application scope

To develop new functionalities in SNow, the best practice is to utilize an application scope. This approach allows for the definition and management of an application's boundaries, ensuring that one application does not interfere with another by controlling access to its components.

The process to request an application scope in AMADEUS is quite laborious, as it requires the completion of a form and the approval of the ServiceNow team. After the request was approved, an application scope named "acrs-sre" was created and assigned to us.

Within an application scope is possible to create tables, workflows, actions, dashboards and much more. In chapter 8 a workflow will be deployed within this new application scope.

A "workflow" in ServiceNow is the automation of a multistep process that uses inputs from an alert or another workflow to produce an output result. Workflows in ServiceNow are designed to streamline and automate processes, ensuring efficiency and consistency in operations. If there is no need to set up a specific trigger, a subflow could be defined. It can still be explicitely called in other contexts like in an alert management rule. Another type of configurable operation is the "action". Action, flows and integrations of an application scope are grouped into a "spoke". If they are public it is possible to use them in other scopes. While some spokes are core functions of SNow, others are custom-built or can be bought from the store.

Regarding the instances, DEV environment will be used to develop and test a workflow in part III. Also PROD environment will be used in order to test a real use case.

4.2.2 Win@proach

For the sake of completeness, it is important to mention Win@proach, the legacy corporate application that will remain in use until Q4 2025 at AMADEUS. This tool supports the creation of tickets for change management, Problem Tracking Records (PTR in test or production), Incident Records (IR) and Software Promotion. The data collected from WIN@proach can be used to generate major and minor reports that assist in decision making.

The transition to SNow is ongoing, with all entries created in SNow being replicated to win@proach, but not vice versa. For the purpose of the thesis, the focus will primarly be on SNow as it is the tool that will be supported in the long term.

4.3 Automation tools

The main tools used in the company in order to perform automations are AWX and Jenkins.

4.3.1 AWX

AWX is an open-source project that offers a web-based user interface, REST API, and task engine for Ansible, designed to facilitate the management and automation of IT infrastructure and applications. Serving as the upstream project for Ansible Tower, a commercial offering from Red Hat, AWX aims at providing a robust and flexible platform for managing Ansible playbooks, inventories, and job scheduling.

With AWX, users can create and manage projects containing Ansible playbooks, define host inventories, and set up credentials for accessing those hosts. The platform supports job scheduling, allowing tasks to be run on-demand or at specified times. Additionally, AWX offers detailed logging and reporting capabilities, simplifying the tracking of playbook execution and troubleshooting of any issues.

A key feature of AWX is its role-based access control (RBAC), which enables administrators to define permissions for different users and teams, ensuring that only authorized personnel can execute specific tasks or access certain resources. AWX also supports integration with various authentication providers, such as LDAP and OAuth, facilitating seamless integration with existing user management systems. This is fundamental for granting access to specific service accounts.

Within AMADEUS many standard operations in CouchBase can be performed using Ansible playbooks, making AWX an ideal tool for automating these tasks through its REST APIs.

4.3.2 Jenkins

Jenkins is an open-source automation server that facilitates the continuous integration and continuous delivery (CI/CD) of software projects. Originating from the Hudson project, Jenkins has become a cornerstone in modern DevOps practices, enabling developers to automate various stages of their software development lifecycle. By providing a robust platform for building, testing, and deploying applications, Jenkins helps teams to deliver high-quality software more efficiently.

At its core, Jenkins operates through a series of jobs or pipelines, which define the steps required to build and deploy software. These pipelines can be configured to trigger automatically based on specific events, such as code commits or pull requests, ensuring that new changes are continuously integrated and tested. This automation reduces the manual effort required for repetitive tasks, allowing developers to focus on writing code and improving their applications. Also for Jenkins it is possible to trigger them with REST APIs.

Jenkins supports a wide range of plugins, extending its capabilities to integrate with various tools and technologies used in software development. This extensibility makes Jenkins highly adaptable to different project requirements and workflows. Whether it's integrating with version control systems like Git, running automated tests, or deploying applications to cloud environments, Jenkins provides the flexibility needed to streamline the entire development process.

By automating the CI/CD pipeline, it helps teams to deliver software faster and with greater reliability, ultimately enhancing the overall quality and efficiency of their development processes.

In AMADEUS' company there are different Jenkins pipeline, dedicated to different purposes. In particular the SRE team has already a set of jobs that could be used in an auto remediation context, so it is in our interest support also this tool in the self-healing mechanism.

4.4 Tools in AMADEUS

Given the established concept of self-healing, it was anticipated that some tools would already exist within AMADEUS. Research across different teams revealed two tools: "Recovery Orchestrator and Switch Automation" (ROSA) and "Auto Remediation of Alerts."

4.4.1 Recovery Orchestrator and Switch Automation

One of the relevant tool for this study is (ROSA). This self-service tool allows specific teams to manually trigger remediations in case of incidents, which may impact a single application, a single PaaS, or an entire region/datacenter. Different actions are taken depending on the case. ROSA primarily focuses on cloud-native resources, with orchestration realized in ServiceNow, while actual recovery actions occur in AWX, manually. The process involves a central Bitbucket repository for recovery scripts, a Jenkins CI/CD pipeline for promoting new or updated scripts, and the ServiceNow UI for Front-Line-Service (FLS).

4.4.2 Auto Remediation of Alerts

SNow's team developed a fully self-healing tool to react in case of ARGOS firing alerts. The tool is designed to attempt a single remediation in AWX Tower, and only in case of failure to create an Incident Record (IR). It provides a perfect base line to test simple cases of self-healing therefore it will analyzes in chapter 7.

4.5 Conclusion

With a comprehensive understanding of technologies and tools utilized within AMADEUS, it is now possible to evaluate their integration into a self-healing mechanism. This will be the primary objective of Part III.

Monitoring Tools: ARGOS is the principal monitoring tool used in the company, so the main focus will be on it. However, unique logs from Splunk will also be integrated as necessary.

Incident Management Tool: Only ServiceNow will be used, as Win@proach is scheduled for decommissioning by the end of the year.

Automation Tools: Both AWX and Jenkins will be employed, as they offer valuable functionalities to the team.

Auto Remediation of Alerts: this tool provides a true self-healing mechanism by triggering auto-remediation in AWX upon the occurrence of an alert originating from ARGOS. It will be thoroughly explained, upgraded, and adapted to our CouchBase use case in Chapter 7.

Although not fully self-healing as it requires manual triggering, ROSA served as a valuable reference for orchestration and automation. This combined with the auto remediation of alerts, will lead to the development of a SRE-specific tool in Chapter 8.

With the technical background established, particularly for Part III, we can proceed to Part II, which will analyze the built-in self-healing mechanisms of various applications used within AMADEUS.

Part II

Self-Healing in cloud applications

Chapter 5

Built-in mechanisms

This chapter provides an overview of the available out-of-the-box self-healing mechanisms in various applications and technologies.

5.1 Couchbase

Couchbase (CB) is a distributed NoSQL database platform designed for high performance, scalability, and flexibility, and offers numerous self-healing features.

A Couchbase cluster comprises one or more instances of Couchbase Server, each running on an independent node. These nodes collaborate to store and manage data, which is automatically distributed across the cluster. The nodes are capable of self-monitoring the cluster's state and can take actions to ensure continuous availability without losing data.

The data is stored inside data buckets (essentially a table), which are divided into 1024 virtual buckets (vBuckets). Those vBucket are evenly distributed across all nodes and can be replicated across different nodes. Different services (such as data, query, and indexing) can be distributed based on workload.

Each node contains a Couchbase Cluster Manager that oversees communications between nodes and ensures their health. Nodes can be added or removed from the cluster as needed, with data automatically rebalanced across the cluster to maintain optimal performance[5].

There are different ways of replication in Couchbase. Intra-cluster replication ensures data is copied within the same cluster across multiple nodes. Meanwhile, Cross Data Center Replication allows data to be replicated between different clusters, often located in separate data centers, to support disaster recovery and geographical distribution.

Couchbase's self-healing features ensure continuous availability and data integrity. One primary mechanism is Cluster Health Monitoring, which allows the main node to monitor the status of other nodes within the cluster. However, certain operations are not performed automatically during failover, such as rebalancing the cluster or reintegrating the node if it comes back online.

Intra-cluster Replication

Intra-cluster replication further enhances data resilience by replicating data across multiple nodes within the same cluster, providing redundancy and quick recovery options in case of node failures. Each vBucket has an active instance and can have up to three replica instances, which are used solely for redundancy and kept in sync by the Database

Change Protocol (DCP). The cluster manager can promote a replica vBucket to become the new active vBucket in case of failure, ensuring the optimal distribution of vBuckets to minimize the risk of data loss, and ensuring that no replica resides at the same node as its active counterpart **??**. An example of intra-cluster replication is shown in Figure 5.1.

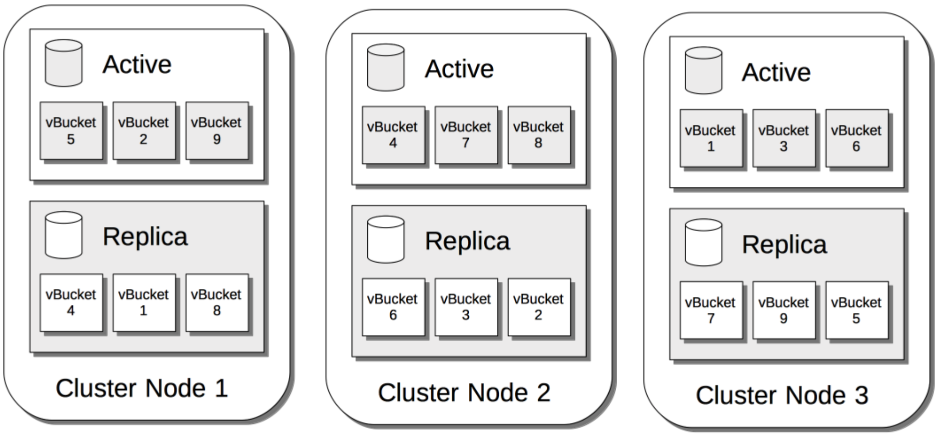


Figure 5.1: Intra-cluster replication in Couchbase[7]

For intra-cluster replication it is possible to set up the auto failover mechanism. It detects node(s) failures or disk r/w failure and automatically transfers the workload to a standby node, minimizing downtime and ensuring operational continuity. Automatic failover is limited to one node at a time and can be configured to occur up to three times. Moreover, it works exclusively for intra-cluster replication.

Cross Data Center Replication

Cross Data Center Replication (XDCR) is another critical feature in Couchbase. XDCR enables data replication across different data centers, ensuring data accessibility and consistency even in the event of a complete data center failure. This feature is particularly valuable for disaster recovery and maintaining high availability across geographically distributed environments[6].

XDCR replicates data from a source bucket in one cluster to a target bucket in another cluster, with both clusters potentially located in different geographical regions. This continuous propagation of mutations ensures that the target bucket remains up-to-date. XDCR is beneficial for disaster recovery and data locality, allowing for the configuration of replication direction and topology, either unidirectionally (active/standby) or bidirectionally (active/active). An example of intra-cluster replication is shown in Figure 5.1.

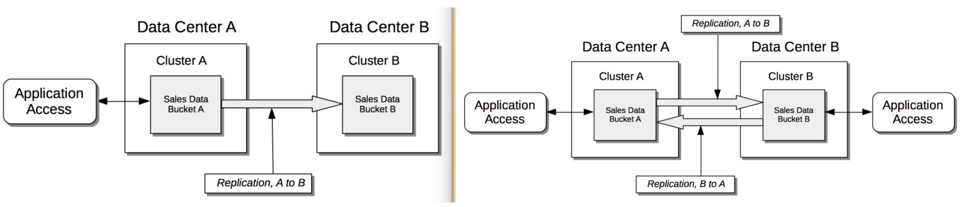


Figure 5.2: XDCR in Couchbase (unidirectional — bidirectional)[8]

It is possible to conclude that CB has already some self-healing mechanism like the auto-failover feature which contributes to ensure high availability and quickly recover from a node failure. However, some features were not implemented by design, like the automatic rebalancing of the cluster after a failover, or the automatic reintegration of a node after a failure. In order to cover these gaps it is necessary to implement some external mechanism, in particular the reintegration of a node after a failure will be implemented in chapter 7.

5.2 Kubernetes

Transitioning from database management to infrastructural services, the focus now shifts to self-healing mechanisms provided by Kubernetes.

Kubernetes (k8s) is an open-source platform designed to automate the deployment, scale, and manage containerized applications. The entire computational layer in AMADEUS is based on Kubernetes.

Kubernetes is built around the concept of a cluster, which consists of one or more nodes. Each node can host multiple pods, which are the smallest deployable units in Kubernetes. The pods contain one or more containers that share resources and are scheduled together at the same node[18].

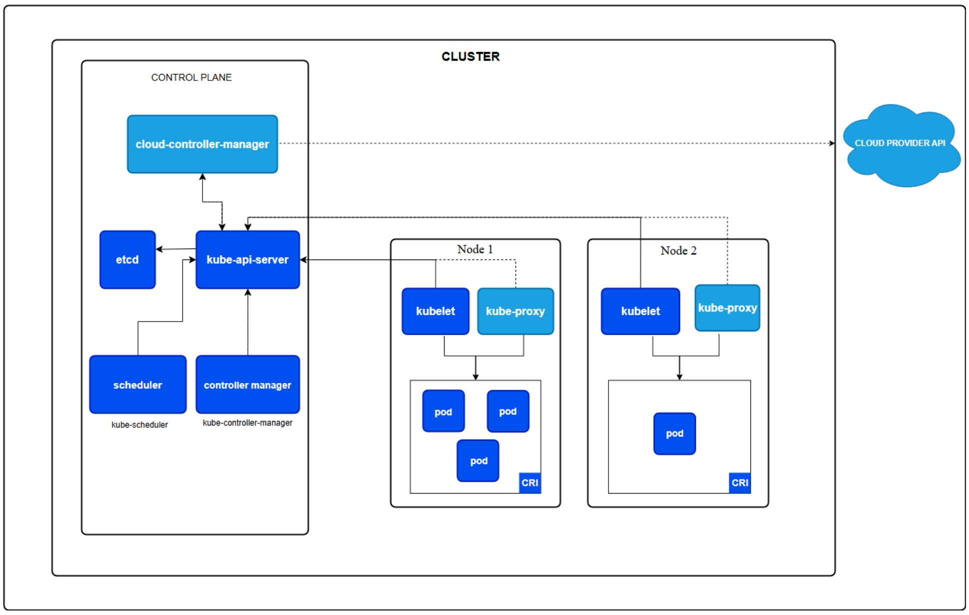


Figure 5.3: Kubernetes cluster architecture[18]

Within k8s, there are different components that ensure self-healing at different levels of the architecture. The objective in k8s more than to resolve the issues is to maintain the desired state of the cluster, usually in terms of the number of pods running or the resources consumed. Natively k8s is capable of monitoring the status of pods and containers. Provides features to replace unhealthy containers according to a configuration file. It can also replace pods in case of malfunctions.[4]

kubelet

The Kubelet component is responsible for managing and coordinating pods and nodes. Its functionalities include pod deployment, resource management, and health monitoring, all of which enhance the operational stability of a Kubernetes cluster[9].

Pod restarting containers

In case a container crashes there are multiple possible actions that the kubelet can take.

If the container crashes for the first time, then K8S attempts an immediate restart based on the pod restartPolicy.
In some cases, due to more several bugs, repeated crashes could occur. In this case, Kubernetes applies an exponential backoff delay for subsequent restarts, always described in restartPolicy. This prevents rapid and repeated restart attempts from overloading the system.
When the container keeps failing and restarting repeatedly, it enters a state called CrashLoopBackOff. This indicates that the backoff delay mechanism is currently in effect. If a container runs successfully for a certain duration (e.g., 10 minutes), Kubernetes resets the backoff delay, treating any new crash as the first[21].

Probes restarting containers

There are three possible container states: Waiting, Running, and Terminated. It is possible to set up three different kind of probes which are liveness, readiness, and startup probes.

Liveness probes determine when to restart a container. It is meant to detect when the application is running but unable to actually serve requests or make progress. For example it could catch a deadlock. Kubelet is in charge of restarting the container following the restart policy if the liveness probe fails repeatedly.

The second kind of probe is the readiness probe. It is used to determine when a container is ready to accept traffic. This is useful when waiting for an application to perform time-consuming initial tasks, such as establishing connections, loading files or data from a DB. While the readiness probe is failing, k8s removes the pod from all matching service endpoints.

Lastly the startup probe objective is to verify if the application within a container is started. While this probe is failing, the other two are disabled since it would not make sense to test the liveness of a container not yet fully initialized. In this way it avoids killing the container before it is fully started. Unlike the other 2 types of probe, the startup probe is executed only at the beginning of the container lifecycle.

Another way to avoid the killing of the probes at startup is to define a "initialDelaySeconds" in the configuration file, however it is clearly less precise[19].

Replacement of pods

Another kind of action performed by kubernetes, in this case performed by the kube-controller-manager, is the replacement of pods (in specific scenarios). When on a node the maximum limit of utilizable resources is reached, the node will perform a node-pressure eviction. The name comes from the fact that the node is under pressure and needs to free up resources.

The control plane (kube-controller-manager) will then create new pods, potentially on a different node, replacing the evicted pods. That happens only for pods managed by a workload management object, such as StatefulSet or Deployment, that replaces failed pods[20].

5.3 Azure Kubernetes Service

The Kubernetes clusters can be hosted on Azure Kubernetes Service (AKS), a managed Kubernetes service provided by Microsoft Azure. There is a small addition that they provide in terms of self-healing mechanisms, which is the AKS node auto-repair feature.

They also provide a set of best practices for designing self-healing applications, which are worth mentioning.

AKS node auto-repair

AKS continuously monitors the health of worker nodes by looking out for specific issues. One key indicator is the NotReady status, which is flagged if a node shows this status during consecutive checks within a 10-minute period. Another important sign is the absence of any status report from a node within the same timeframe.

When AKS identifies an unhealthy node that remains in this state for at least five minutes, it initiates a series of repair actions. The first step is to reboot the node, as this often resolves transient issues. However, if the node continues to be unhealthy after a reboot, AKS takes a more drastic measure by reimaging the node. This process involves reinstalling the operating system and resetting the node to its original state. For Linux nodes, if reimaging does not resolve the issue, AKS redeploys the node by creating a new instance and replacing the unhealthy one.

AKS employs a retry mechanism, where it repeats the sequence of reboot, reimage, and redeploy up to three times if the node remains unhealthy. This entire auto-repair process can take up to an hour to complete. Throughout this process, AKS generates Kubernetes events from the aks-auto-repair source, providing visibility into the actions taken. Administrators can monitor these events to track the status of node repairs and take additional actions if necessary.

It is important to note that node auto-repair is a best-effort service and does not guarantee that the node will be restored to a healthy state. In some cases, manual intervention may be required. Additionally, auto-repair may not be performed in certain scenarios, such as network configuration errors or during node upgrades. Despite these limitations, this process helps ensure that your AKS cluster remains healthy and minimizes service disruptions by automatically addressing node issues[23].

Recommended Designs

Azure defines some design principles that should be implemented in order to develop a self-healing system[25]. They are mostly at the application level, so not in the scope of the SRE team, however it is interesting to mention a couple of them.

Retry logic: Implementing retry mechanisms for transient failures, such as momentary network issues or database connection drops, is crucial. Many Azure services provide automatic retries through their client SDKs.

Failover Mechanisms: For stateless components, deploying multiple instances behind a load balancer ensures availability. For stateful components, using replicas and failover strategies can help maintain consistency and availability.

Decoupled Components: Components should communicate asynchronously to minimize the risk of cascading failures. This can be achieved through event-driven communication, ensuring that components do not need to be present simultaneously or within the same process.

Circuit Breaker Pattern: To prevent persistent failures from overwhelming a system, the Circuit Breaker pattern can be employed to fail fast when an operation is likely to fail, thereby avoiding excessive load on failing services.

A really interesting concept is chaos engineering, which involves randomly injecting failures into production. This methodology can be used to test the reliability of the system and eventually to test self-healing mechanisms. It could also be used to simulate the failure of components in order to train an AI.

5.4 Conclusion

This chapter showcased how self-healing mechanisms are implemented out of the box in some technologies. Couchbase and Kubernetes two key technologies used at AMADEUS were taken as a case study.

The structure seen in this case is the same, there is a monitoring component, an orchestrator that decides how to react, and a reactor that will remediate the situation.

These technologies do not cover every possible need of the SRE team, and in some cases, they need to be integrated with external tools. The next part will focus on externally implemented self-healing mechanisms.

Part III

Self-Healing through monitoring

Chapter 6

Introduction

As outlined in Part I, current sensors retrieve metrics from various applications. These metrics are primarily utilized to send alerts, requiring manual intervention for any subsequent actions.

This chapter presents a more proactive approach by discussing a framework that implements self-healing mechanisms. These mechanisms are not managed within the application itself but are instead triggered by an external application.

The concept of self-healing can be divided into three components: the initial step involves detecting the issue, either through logs or metrics. Subsequently, a central orchestrator is needed to recognize the issue and initiate a remediation action. Finally, a reactor is necessary to execute the remediation action.

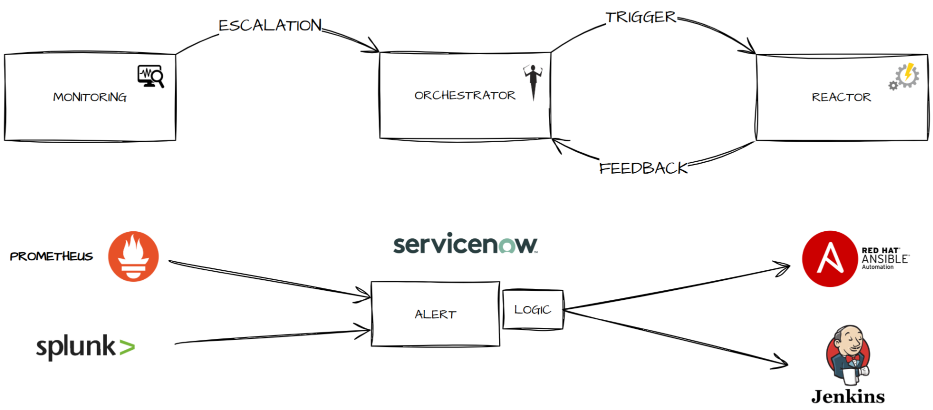


Figure 6.1: High-Level Design

Several design considerations will be addressed:

- Sensoring Scope: It is necessary to determine whether to limit the scope to metrics or to include comprehensive log management. The potential additional benefits of full log management in the context of cloud self-healing will be evaluated.
- Research Tools: Existing tools will be assessed, and their implementation will be considered from design to execution.
- Technology: The appropriate technology, types of input/output, and the self-healing capabilities of the framework will be identified.
- Decision Engine: Various approaches, such as rule-based systems, artificial intelligence, and human decision-making via a user interface, will be considered.

Key factors to be determined include:

- Feasibility: The solution's feasibility within the context of AMADEUS will be assessed.
- Reliability: The proposed solution must effectively resolve the problem and should not introduce additional issues.
- Cost: This encompasses storage and computation costs, as well as licensing fees if external tools are utilized.

Known issues can be resolved either within the technology itself, as discussed in the previous chapter, or through configuration for self-healing using external tools. This part primarily focuses on addressing these issues using ServiceNow as an orchestrator and a rule-based system. Specifically, the existing "auto remediation of alerts" tool will be analyzed and utilized in chapter 7. It will cover simple cases that integrates with ARGOS and AWX.
An improved application, tailored for the SRE team will be developed in chapter 8. This second application will tackle more complex incidents and integrate with more technologies.

Chapter 7

”Auto Remediation of Alerts” - simple use cases

”Auto remediation of alerts” is one of the tools developed by the ServiceNow team at AMADEUS to provide self-healing in response to firing alerts. This chapter will analyze the tool and its limitations, propose minor improvements, and implement a use case in CouchBase for the team.

The tool, implemented as a SNow workflow, is not widely used due to its recent introduction. Only a few rules are configured, and some features are missing.

The workflow is designed to address alerts fired from ARGOS by attempting remediation in AWX Tower, rather than directly creating an Incident Record (IR). If the remediation is unsuccessful, the standard procedure to generate the IR continues.

The rationale behind this tool can be understood from its design. By preventing the creation of an incident if the remediation is successful, the primary objective is likely to reduce noise in the incident system, thereby decreasing the need for manual intervention. However, a potential drawback is that a misconfiguration or bug may go unnoticed, as the incident is no longer created.

The most suitable alerts for configuring rules in this tool are those known to be easily fixed, with well-defined solutions or repetitive operations. Ideally, the solution should not involve delicate steps that could further damage the system

Some example use cases could be:

- Reintegrating a CouchBase node after a failover. (the issue explained in chapter 5)
- Janitoring evicted K8s resources.
- Detection and renewal of expiring encryption keys.

In this chapter, the implementation of the tool will be explored in detail to understand the processes occurring behind the scenes. Subsequently, the steps required to configure a new rule—from ARGOS to ServiceNow and then to AWX—will be examined. Following this, the tool’s limitations will be discussed. Finally, a minor improvement to the tool will be proposed, and the previously mentioned CouchBase use case will be implemented.

7.1 Monitoring: ARGOS Escalation

As previously mentioned, ARGOS is the proprietary monitoring framework of AMADEUS, based on Prometheus. This framework allows for the setup of escalations to various tools, including ServiceNow (SNow).

Since there are multiple SNow instances, the escalation is designed to target only the relevant instances, preventing unnecessary flooding. A schema of the escalation links is depicted in Figure 7.1. To identify the production (PRD) platforms, a Grafana dashboard that monitors the state of the forwarders can be consulted. For the SRE team, even platforms internally referred to as development (DEV) or test (TST) are considered PRD platforms by SNow.

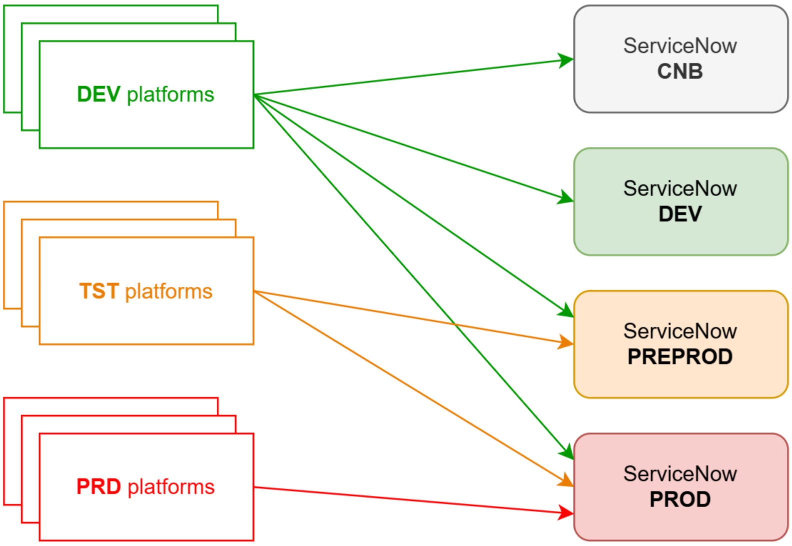


Figure 7.1: ARGOS Snow Forwarder

Once the event is parsed in SNow and an alert is created, it requires specific parameters to categorize the error and identify its origin. These parameters are configured in the ARGOS labels and include:

- **servicenow_event_type:** This parameter specifies the type of event that triggers the alert, such as DB_ERROR, APP_ERROR, or NETWORK_ERROR.
- **servicenow_severity:** This parameter is a numerical value from 1 to 4 that indicates the severity of the alert, ranging from critical (1) to warning (4). Alerts set to warning will not generate an incident.
- **servicenow_ci_type:** This parameter identifies the type of Configuration Item (CI) on which the issue is found. It must correspond to one of the CI types configured in the CI model, such as a Kubernetes (K8) cluster or an F5 pool.
- **servicenow_ci_tag:** This optional label specifies a unique tag to identify the CI. It may be required in cases where the SNow CI binding process cannot correctly identify the CI.

In addition to these labels, other information defined by the user may be passed as annotations. The difference between the labels and the annotation is that the labels will be parsed by SNow and eventually used for specific scope, while the annotations are fully customizable. The fields specified in the annotations will be stored into the "additional_info" field of the SNow event and alert.

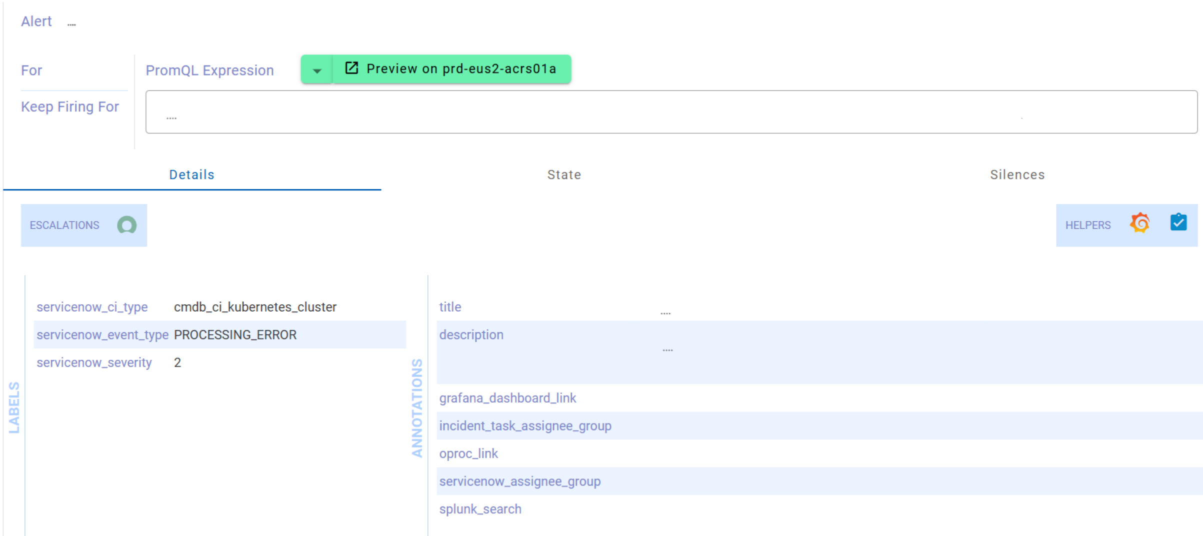


Figure 7.2: ARGOS Escalation

7.2 Automation: AWX template

The remediation action triggered by the SNow workflow is an AWX template. It is necessary to set up the template in AWX before linking it to the SNow configuration table. It has to be tested extensively in order to avoid the creation of undesired issues.

The template can be either a job template or a workflow template. Once created, only a few additional steps are required to make the template triggerable by SNow.

First, the ServiceNow robotic user needs the access permissions to execute the template. This can be achieved by configuring the appropriate roles and permissions within the AWX template as shown in Figure 7.3.

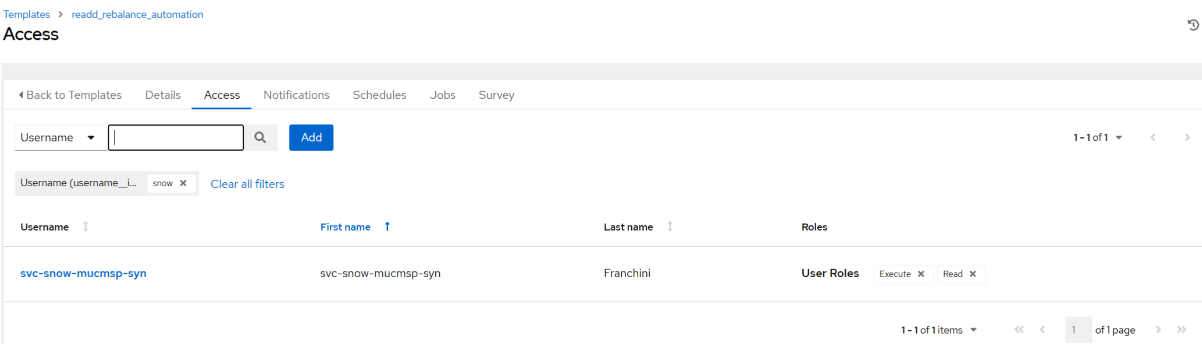


Figure 7.3: AWX Configuration 1

To enable the automation to pass variables dynamically at runtime, it is essential to check the "prompt at launch" box for the extra-vars. These extra-vars, passed from SNow to AWX, are contained in the "additional_info" field of the SNow event, which includes all the annotations set up in the ARGOS alert. Even if there are no variables to pass, this box must be checked. Without this configuration, the automation will be triggered, but ServiceNow will not receive any updates on its status. Consequently, without feedback, it will be unable to determine whether the template has completed.

Additionally, if it is necessary to pass the limit dynamically and a limit annotation is set up in the ARGOS alert, the limit's "prompt at launch" box must also be checked. This ensures that the automation respects the specified limits and operates within the defined scope.

Finally, the ID of the template, which can be found in the URL of the template's page, should be saved. All these changes can be made from the AWX UI as shown in Figure 7.4.

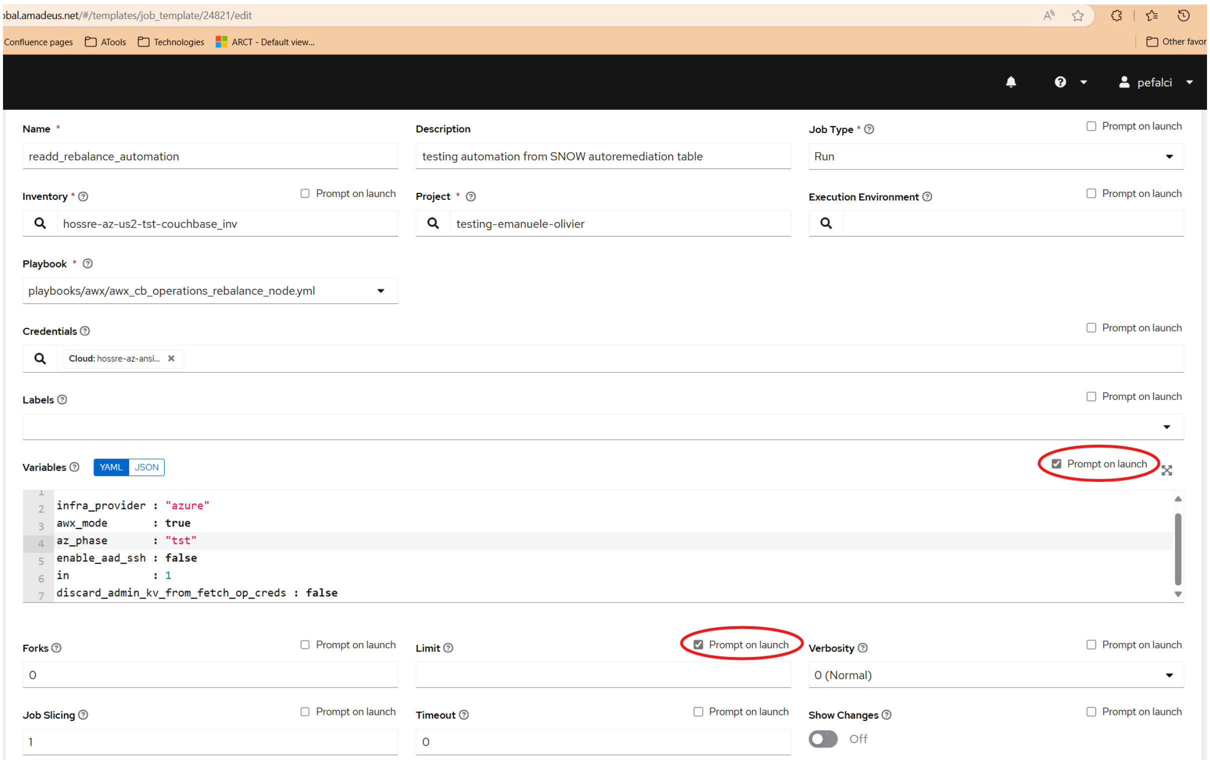


Figure 7.4: AWX Configuration 2

7.3 Orchestration: SNow

The entire orchestration process occurs within SNow. Upon receiving the event from ARGOS, it is processed according to the defined event rules, resulting in the creation of an alert with the appropriate fields. If the alert continues to fire and another event is generated, it will be linked to the same alert to prevent the creation of multiple alerts for the same issue. During this process, SNow will also attempt to perform Configuration Item (CI) binding, linking the event to the correct affected CI.

Once the SNow alert with an ARGOS source is created, it will match an alert management rule that passes it as input to a SNow subflow called "alert auto-remediation". 7.5 Within this subflow, the actual automation is triggered if the correct requirements are met, the first of which is the presence of a record in a configuration table (described in the next subsection) for that alert.

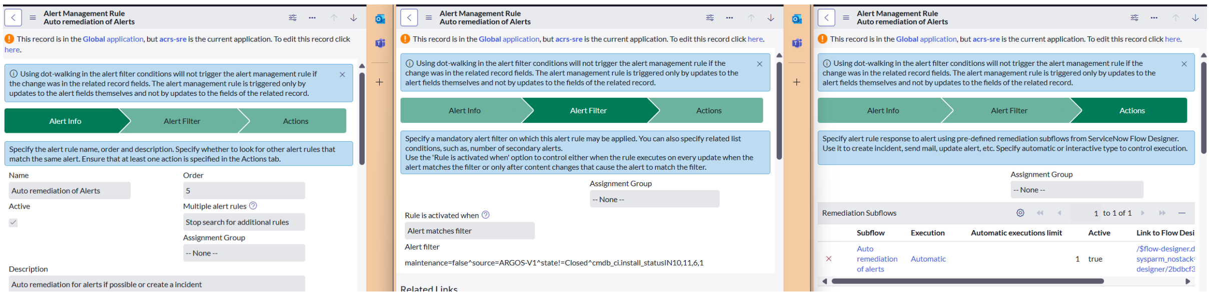


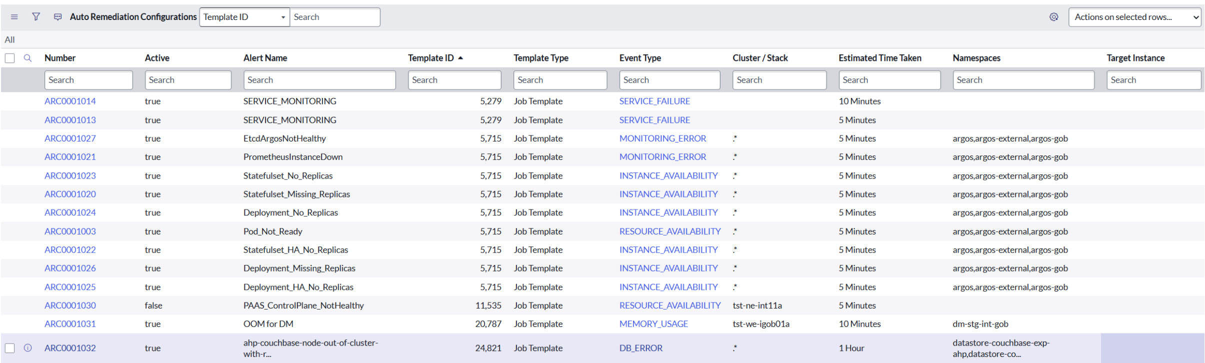
Figure 7.5: Alert Management Rule

Assuming all configurations are correctly set, the subflow initiates the AWX remediation, which may involve either a template or a workflow. The process includes passing all variables contained in the "additional_info" field.

If the AWX template successfully resolves the issue, the Incident Record (IR) is not created, and the alert is closed. Conversely, if the template fails or times out, an IR is generated to ensure manual resolution of the issue. This methodology reduces the need for manual interventions and ensures that only unresolved issues escalate to the incident management process.

7.3.1 SNow Configuration Table

The configuration table is essential for establishing auto-remediation actions, as it stores a one to one link between any alert and its remediation action. These parameters collectively ensure that the auto-remediation rule is accurately configured and effectively managed. A detailed explanation of the parameters involved is provided in Table 7.1 and the full table is shown in Figure 7.6.



Auto Remediation Configurations

Number	Active	Alert Name	Template ID	Template Type	Event Type	Cluster / Stack	Estimated Time Taken	Namespaces	Target Instance
ARC0001014	true	SERVICE_MONITORING	5,279	Job Template	SERVICE_FAILURE		10 Minutes		
ARC0001013	true	SERVICE_MONITORING	5,279	Job Template	SERVICE_FAILURE		5 Minutes		
ARC0001027	true	EtcdArgosNotHealthy	5,715	Job Template	MONITORING_ERROR	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001021	true	PrometheusInstanceDown	5,715	Job Template	MONITORING_ERROR	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001023	true	Statefulset_No_Replicas	5,715	Job Template	INSTANCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001020	true	Statefulset_Missing_Replicas	5,715	Job Template	INSTANCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001024	true	Deployment_No_Replicas	5,715	Job Template	INSTANCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001003	true	Pod_Not_Ready	5,715	Job Template	RESOURCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001022	true	Statefulset_HA_No_Replicas	5,715	Job Template	INSTANCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001026	true	Deployment_Missing_Replicas	5,715	Job Template	INSTANCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001025	true	Deployment_HA_No_Replicas	5,715	Job Template	INSTANCE_AVAILABILITY	.*	5 Minutes	argos,argos-external,argos-gob	
ARC0001030	false	PAAS_ControlPlane_NotHealthy	11,535	Job Template	RESOURCE_AVAILABILITY	tst-ne-int11a	5 Minutes		
ARC0001031	true	OOM for DM	20,787	Job Template	MEMORY_USAGE	tst-we-igob01a	10 Minutes	dm-stg-int-gob	
ARC0001032	true	ahp-couchbase-node-out-of-cluster-with-r...	24,821	Job Template	DB_ERROR	.*	1 Hour	datastore-couchbase-exp-ahp,datastore-co...	

Figure 7.6: Auto Remediation Configuration

7.3.2 SNow Workflow

As previously mentioned, the primary logic operates within a SNow subflow, which is triggered by an alert management rule following the creation of an alert from ARGOS.

The main components of the workflow are as follows:

Parameter	Description
Template ID	The unique identifier for the template to be executed on AWX, ensuring the correct automation script is triggered.
Template Type	Specifies whether the template is a job template or a workflow template. Job templates are typically used for single tasks, while workflow templates handle more complex sequences of tasks.
Alert Name	The name of the alert from ARGOS that triggers the automation, acting as a key to link the alert with the appropriate remediation action.
Cluster Stack	Indicates where the application is deployed.
Namespace	Copied from the namespace specified in the PromQL expression. It helps scope the alert to a specific part of the infrastructure.
Estimated Time Taken	Sets a threshold for the duration of the playbook's execution. If the playbook exceeds this time, an Incident Record (IR) is created to ensure prolonged issues are addressed promptly.

Table 7.1: ServiceNow Configuration Table Parameters

- Search in a configuration table the record with the matching name of the received alert. If found, it further verifies that other parameters, such as namespace and event type, also match.
- Updating the alert work notes in ServiceNow to indicate the initiation or non-initiation of the remediation action.
- Initiating the AWX job template or workflow by calling the REST API through a MID server.
- Sending an email to the assigned group, if necessary.
- Upon completion of the job, if it fails, updating the alert work notes with the AWX output and creating an Incident Record (IR). If the job succeeds and the alert ceases to fire from ARGOS, the flow concludes successfully. However, if the alert continues to fire, an IR is created regardless.

In case of need it is possible to see the logs of a specific exection of the flow. This might be useful in case the auto remediation is not triggered correctly or there are other issues at the orchestration level.

Understanding the connection to AWX is essential. The workflow employs utilities provided by the Ansible Spoke, enabling the creation of a record within a table called "Ansible Jobs". Another workflow is triggered whenever a record is created or modified in this table, which subsequently calls REST APIs to trigger the AWX template using a SNow robotic account's credentials. In order to do so, since SNow and AWX are not in the same network, they need to be connected through a MID server.

Receiving a response from AWX is also crucial in order to have a feedback of the execution. As the template is not executed synchronously, it is not possible to obtain the response directly from the REST API call. Instead, the response is sent back from AWX upon completion of the template's execution and is stored in the integration transaction table. Another workflow modifies the state of the Ansible record, transitioning it from "running" to "closed completed" or "closed incomplete".

The original SNow workflow will wait until the job status is no longer "running" and then retrieve the job output from the integration transaction table to print it in the alert's work notes.

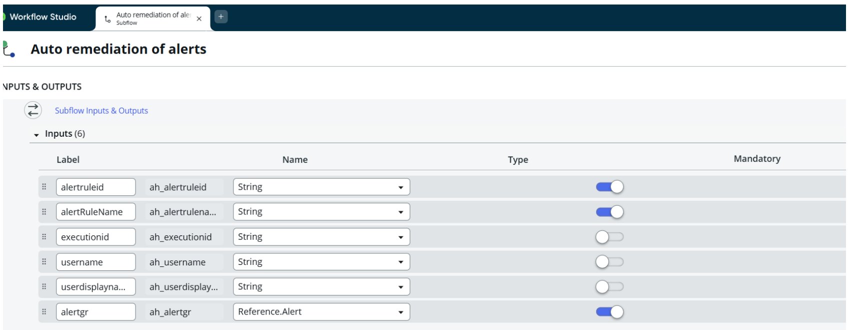


Figure 7.7: Auto Remediation of Alerts

7.4 Limits and improvements

This method of auto-remediation presents several advantages and limitations.

As previously discussed, the tool is straightforward to set up if an alert rule and a remediation action are already in place. When used for non-critical operations that still require time and human intervention, it can significantly reduce the team's workload and enhance overall efficiency.

However, several limitations can be identified. For more complex incidents, the tool may be unable to react effectively. There is no retry mechanism in case of failure, and the configuration table is one-to-one, preventing the definition of multiple possible solutions for a common problem. Additionally, there is no provision for a human approval step before remediation in cases of testing or potentially destructive operations.

Regarding integration with other tools, the workflow works with ARGOS, the primary monitoring tool. While this is sufficient for the most common use cases, as illustrated in Figure 4.2, there is no disadvantage in also supporting Splunk alerts, which could be beneficial for other use cases. For the automation component, the tool currently supports only AWX. The (SRE) team has numerous automations already built in Jenkins, and it would be advantageous to have a method to trigger these from SNow as well. A workaround could involve triggering a Jenkins job from an AWX template, but this is not an optimal solution.

At present, the tool is unable to handle the playbook's limit. Additionally, there is no standardized method for creating new records in the table, which poses a limitation for the SRE team, as they must request the tool owner to perform this task.

7.4.1 Improvement: ansible limit

One of the primary use cases identified involves executing an AWX job with a specified limit. A simple workaround would be creating a workflow template in AWX, where an initial playbook extracts the limit parameter from the additional_info and then calls a subsequent playbook with the limit set. However, a more streamlined solution was preferred. After consulting the SNow team, as the REST APIs already support passing the limit parameter, a solution was developed to pass it directly from SNow.

Various options were considered for the setup, and it was decided to pass the limit as an annotation with the key ”limit” in the ARGOS alert. In this way there is no risk to mess around with the labels key values that are used by SNow to categorize the alert.

The SNow workflow then checks for the existence of this ”limit” annotation and, if present, includes it in the REST API call to the AWX template. The code is listed below.

```
...
var util = new x_amfr_ansible_amd.ansibleUtil();

var body = {"additional_info": fd_data.flow_var.alert_additional_information}
var limit = JSON.parse(fd_data.flow_var.alert_additional_information).limit

if (limit != null ) {
    //set the outside_extra_vars
    outside_extra_vars = {"limit" : limit}
    body["outside_extra_vars"] = outside_extra_vars
}

var ANSrecord = util.launchTemplate(fd_data.subflow_inputs.ah_alertgr , fd_data.flow_
...
```

The launchTemplate function creates the record in the Ansible Jobs table with the specified fields.

7.5 Couchbase use case

An important use case for implementing this tool can be found in CouchBase. CB does not natively support the re-adding of a node back to the cluster in the event of a failover. Currently, this operation is performed manually, and since nodes often fail over due to non-critical reasons, it results in a significant time expenditure for the Site Reliability Engineering (SRE) team. Since this is a straightforward operation with low associated risks, it is an ideal use case to implement.

This section will provide a detailed explanation of how to configure this use case within the tool, including an in-depth analysis of the metrics used and the remediation implemented.

7.5.1 ARGOS alert

The first step is to establish the alert rule to detect when a node is out of the cluster. By examining the various predefined metrics, the following metrics were identified as relevant for this purpose:

`couchbase_clusterMembership` and `couchbase_operation_rebalance`

To identify CouchBase nodes that are out of the cluster and not currently rebalancing, a simple PromQL expression can be written:

```
couchbase_clusterMembership{cluster=~".*<CLUSTER_NAME>"} == 1
```

and

```
ON (id, cluster, phase, peak) couchbase_operation_rebalance{cluster=~".*<CLUSTER_
```

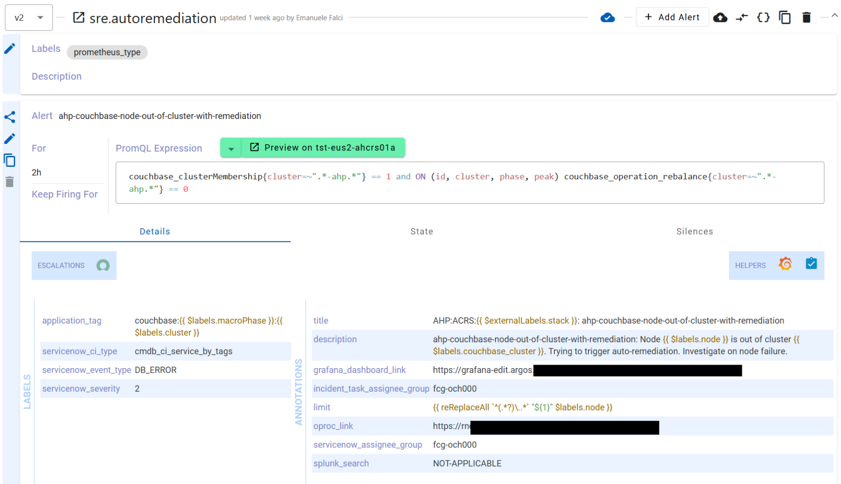


Figure 7.8: ARGOS

Particular attention must be given to the timing of the alert. While triggering the alert as soon as possible is the ideal scenario in order to minimize the time the node is out of the cluster, triggering the alert immediately is not advisable in this specific case. Various factors, such as a virtual machine restart in Azure that will recover automatically, or other (SRE) operations on the cluster, including operating system updates or patching, could cause temporary failures. To avoid interference with regular SRE maintenance operations, the alert is configured to fire only after pending for two hours.

7.5.2 AWX template

The AWX template serves as the remediation action to reintegrate the node into the cluster and execute a rebalance operation.

To achieve this, a playbook from the CouchBase team's repository was adapted to meet specific needs. The playbook, detailed in the appendix, performs standard operations to retrieve the necessary artifacts and passwords. It utilizes the AWX limit parameter to restrict the operation to the targeted node, with an explicit check to ensure the limit is set. Subsequently, the node is re-added to the cluster and rebalanced.

To set up the playbook, a new project was created in AWX and linked to a personal repository on BitBucket where the playbook was uploaded. The playbook, which employs Python scripts, requires a virtual environment with Python installed. Credentials were configured to log into Azure, where infrastructure information, including passwords, is stored. For the inventory, a specific inventory for the test CouchBase cluster was used, as the current focus is on testing servers.

It should be noted that there is also an inventory for production (PRD) servers, but not one that includes both. If the playbook needs to support both phases, a new virtual inventory encompassing both can be created and used for the playbook.

Regarding the AWX template configuration, the "prompt at launch" box for both the extra-vars and the limit parameter were set as can be seen from Figure 7.9. Access was granted to the SNow user to run the template, and the template ID was saved for the configuration in the SNow table.

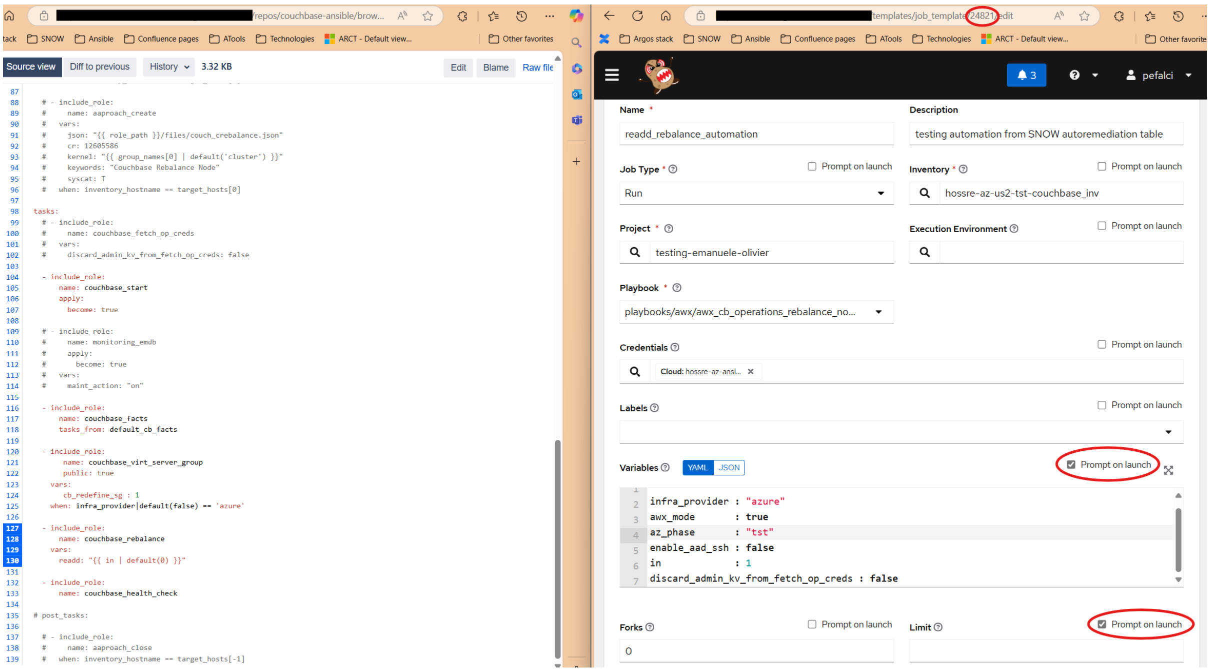


Figure 7.9: AWX Configuration for Couchbase Remediation

7.5.3 SNow configuration table

The final step involves setting up the rule in the configuration table in SNow to link the alert to the corresponding remediation action. The configuration is illustrated in Figure 7.10. As previously mentioned, inserting this record into the table required contacting the owner of the tool.

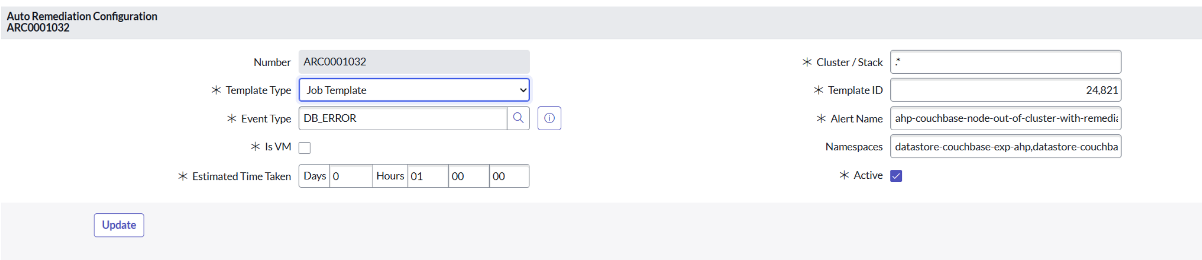


Figure 7.10: Record in the Configuration Table

The configuration parameters are as follows: The **alert name** is set in ARGOS as "ahp-couchbase-node-out-of-cluster-with-remediation." **Namespaces** include the list of namespaces from which the alert could be triggered. To obtain a comprehensive list, all potential namespaces were reviewed from Thanos. The **template type** is a Job Template, not a job workflow, and the **Template ID** is the previously saved ID. The **Event Type** is set to DB_ERROR. Determining the **Estimated Time Taken** is more challenging, as the playbook is not executed synchronously, and the duration is not known in advance. To establish a reasonable time, the playbook was run manually, with execution times typically ranging between 15 and 20 minutes. To avoid premature timeouts, the time was set to 2 hours, although it could likely be safely set to 30 minutes. Upon completing this step, the auto-remediation is ready for testing.

7.5.4 Testing

In order to test the automatic remediation works correctly, the following steps were undertaken:

1. Verify that the escalation from ARGOS to ServiceNow (SNow) functions correctly using a dummy alert that always fires.

2. Ensure that SNow correctly triggers the AWX template configuring a rule in the table that links the dummy alert to a dummy AWX template.

3. Test with the real alert and real template simulating an issue by manually removing the node from the cluster and waiting for the required time for the alert to fire. Verify the alert is firing in ARGOS, and search for the corresponding events and alert in SNow. If they are present, check AWX for the triggered template.

4. Confirm that the playbook is executed correctly and the node is re-added to the cluster. If the playbook fails, ensure that an Incident Record (IR) is created.

After some trial and error, the testing was successful and the alert was correctly remediated. The playbook was executed, and the node was re-added to the cluster. The alert ceased to fire, and no IR was created.

For the sake of testing, the time limit of 2 hours was reduces to just 5 minutes with the main purpose of evaluating the best reaction time of the system but also to test faster. Within the orchestration process, two primary sources of delay were identified: the creation of the event and the execution of the workflow, both of which are dependent on the system load. These delays generally remain under one minute. On the remediation side, an additional time overhead may occur if multiple playbooks are running or if some inventory needs to be synchronized. However, this also typically stays under one minute. Consequently, remediation usually commences within a few minutes of the alert being triggered in Argos proving its efficiency.

7.5.5 Troubleshooting

Mistakes are easily made, particularly when frequently testing and altering configurations. To troubleshoot issues, one can review the work notes or, for more complex problems, examine the flow execution. This instance provides comprehensive information about the flow execution, enabling identification of the problem. Typically, the issue is a configuration error, either in the SNOW table — where the rule could be inactive or the namespace incorrect — or in AWX, where it is necessary to verify that the SNOW user has the appropriate permissions and that the "prompt at launch" box is checked.

7.5.6 Pushing to PRD

To deploy the alert to production, it was necessary to transfer the alert from a private ARGOS project folder to the SRE's project folder. While in the personal project it was possible to work directly from the UI, in the SRE project, the alert was formatted as Dashboard as a Code.

Existing functions within the DaaC were partially reused, and the alert was added to a new folder designated for autoremediations. The alert template follows.

```
/templates/couchbase.template.jsonnet
nodeOutOfClusterRemediation(name):: basicAlert.new(
    name=name,
    query='couchbase_clusterMembership{cluster=~".*-ahp.*"} == 1 and ON (id, cluster,
    metrics=['couchbase_clusterMembership'],
    ci_type='cmdb_ci_service_by_tags',
    servicenow_severity='3',
    servicenow_event_type='DB_ERROR',
    description='Node {{ $labels.node }} is out of cluster {{ $labels.couchbase_clust
    ).addLabel('application_tag', 'couchbase:{{ $labels.macroPhase }}:{{ $labels.clus
    .addAnnotation('limit', '{{ reReplaceAll `^(.*?)\\..*` "${1}" $labels.node }}'),
```

The actual alert was included in an sre.autoremediation file, which corresponds to a folder in Argos.

```
/rules/sre/sre.autoremediation.jsonnet
local alerts = [
  container.new('sre.autoremediation', [
    couchbaseAlerts.nodeOutOfClusterRemediation(
      name='alert-remediation-ahp-couchbase-node-out-of-cluster',
    ),
  ]),
];
```

In accordance with the standard procedure, tests were also written for the alert to ensure its proper functionality. After deploying the alert to production (PRD), the whole process was tested once more to confirm its correct operation.

Chapter 8

SRE Remediation workflow - complex use cases

In this chapter, a new tool tailored specifically for the SRE team is implemented. This tool draws inspiration from the previous one and addresses the need of tackling more complex use cases and integrate also with other tools.

Rationale for Creating a New Tool:
it aims to overcome the limitations identified in the previous section. Therefore it integrates also with Splunk for monitoring and Jenkins for remediation. It supports more complex use cases in which a simple try of a remediation is not sufficient.

The creation of a separate tool was also moved by security concerns. In fact given that SRE operations can impact platform stability, it is crucial to restrict the ability to trigger remediations. The existing "auto remediation of alerts" uses the SNow robotic account in order to run the remediation. This means that every developer in SNow, who by default has access to the robotic account, can trigger the remediation.
Moreover the configuration table is shared among all the teams and is modifiable by everyone. This poses a significant security risk, making it preferable to develop a version of the tool with restricted access.

In order to do so a ServiceNow application scope named "acrs-sre" was requested for the SRE team. Within this scope, it is possible to create tables, workflows, and actions, as well as define the roles that can access them.

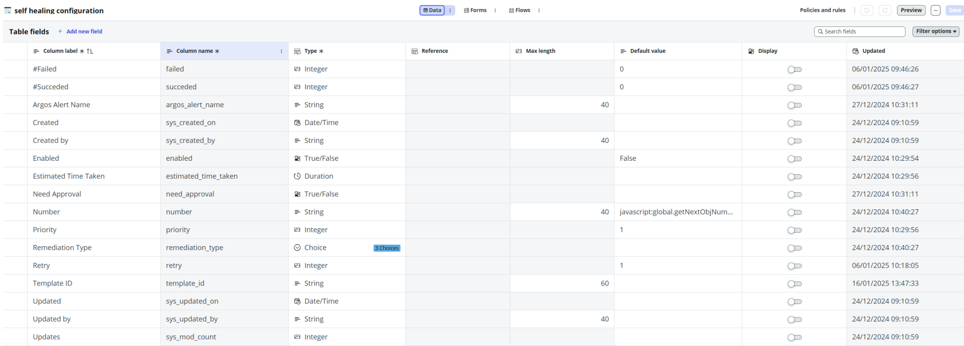
8.1 Design

The "auto remediation of alerts" tool is centered on the auto-remediation of easily fixable alerts, the new application addresses more complex incidents that may not have straightforward solutions. Moreover it supports both ARGOS and Splunk as sources of alerts, and both AWX and Jenkins for remediation actions.

The sre specific application implements, similarly to the "auto remediation of alerts", a rule-based decision system. So the remediations have to be manually configured in order to solve automatically a specific alert, however several key improvements are introduced in the configuration.

The rule-based configuration table will be maintained in a SNow table within the acrs-sre application scope. Shown in Figure 8.1. It includes the necessary fields to link the alert name to the remediation action, along with statistic on the success rate of the rule, a priority field and a retry number.

It includes in fact an opt-in retry mechanism, as it is a typical self-healing design


self healing configuration

Data | Forms | Flows | Policies and rules | Preview | Save

Table fields + Add new field | Search fields | Filter options

Column label	Column name	Type	Reference	Max length	Default value	Display	Updated
#Failed	failed	Integer			0		06/01/2025 09:46:26
#Succeded	succeded	Integer			0		06/01/2025 09:46:27
Argos Alert Name	argos_alert_name	String		40			27/12/2024 10:31:11
Created	sys_created_on	Date/Time					24/12/2024 09:10:59
Created by	sys_created_by	String		40			24/12/2024 09:10:59
Enabled	enabled	True/False			False		24/12/2024 10:29:54
Estimated Time Taken	estimated_time_taken	Duration					24/12/2024 10:29:56
Need Approval	need_approval	True/False					27/12/2024 10:31:11
Number	number	String		40	javascript:global.getNextObjNum...		24/12/2024 10:40:27
Priority	priority	Integer			1		24/12/2024 10:29:56
Remediation Type	remediation_type	Choice 3 Choices					24/12/2024 10:40:27
Retry	retry	Integer			1		06/01/2025 10:18:05
Template ID	template_id	String		60			16/01/2025 13:47:33
Updated	sys_updated_on	Date/Time					24/12/2024 09:10:59
Updated by	sys_updated_by	String		40			24/12/2024 09:10:59
Updates	sys_mod_count	Integer					24/12/2024 09:10:59

Figure 8.1: Rule Configuration Table

pattern. It is possible to set the number of retries, in order to address issues that could be solved just by retrying the same remediation.

To address problems that may have various causes, therefore different possible solutions, it supports multiple possible remediation actions. The order of execution of the remediation actions is chosen thanks to a priority field. At the moment the priority can be set in base of the success rate of the remediation, or using a safe-first approach. In this second case the non-disruptive remediations have highest priority in order to attempt a safer solution first, and only in case of failure try a disruptive one. More complex decision-making mechanisms could be implemented in the future to replace the hard-coded priority, eventually a machine learning model that predicts the best course of action.

Moreover, given that the remediation could be potentially critical, the configuration of the remediation includes an opt-in requirement for human approval. Obviously adding this feature the concept of self-healing is not fully respected as a manual action will still be needed. It represents anyway an improvement in terms of time, as the approval for a disruptive remediation will be asked only safer remediation failed. A positive side effect is that it might come in handy for testing purposes.

In addition, since the target of the new application are more complex cases another modification to the functioning of the "auto remediation of alerts" was made. In fact the existing tool tries to solve the alert even before the creation of an incident. In this case, implementing more complex cases, it is more reasonable having in any case an incident triggered. Doing so, it is easier in the future to analyze how incidents were solved and keep track of them.

Lastly, an additional integration with Teams channel is supported. It is used in order to easily monitor the application, and to be promptly notified in case of need for a human approval.

The workflow

In Figure 8.2 it is possible to see the flowchart of the application that was then implemented in SNow.

Cost analysis

The cost analysis for implementing the SRE Remediation workflow on the ServiceNow platform highlights several key aspects.

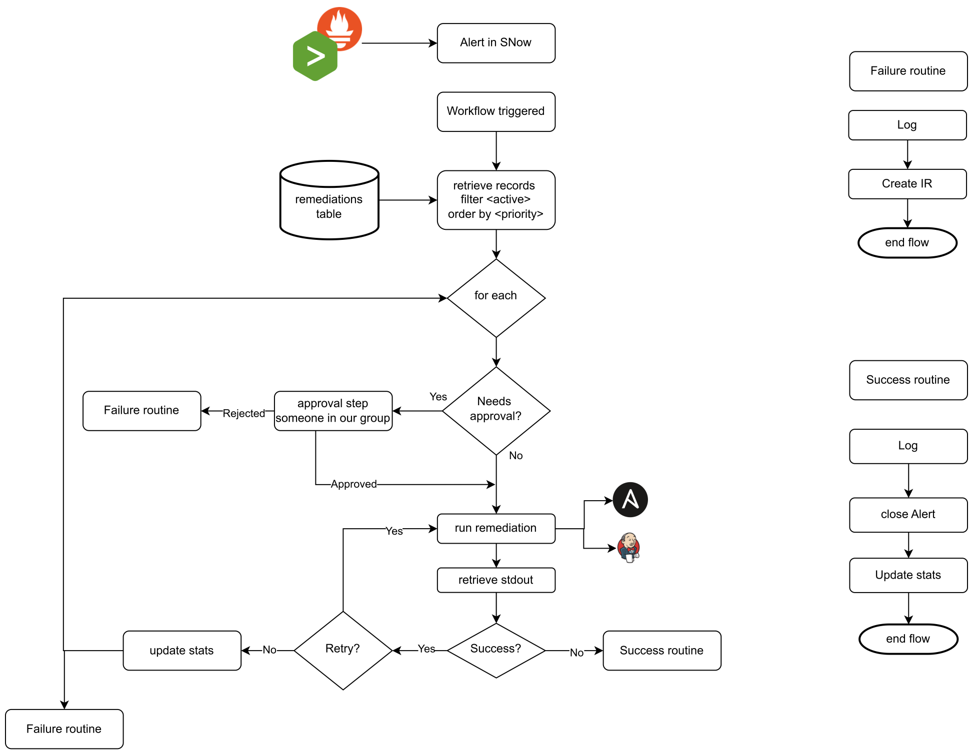


Figure 8.2: Flow chart

Firstly, the time required for the automation to trigger was considered. Since it runs on the same platform, it has the same latency as the previous tool, it is therefore possible to trigger the remediation in a couple of minutes. The total amount of time required to recover the issue depends on the sum of this small overhead and the time required for the remediation action to complete. An additional penalty that should be considered in the case multiple remediations are subsequently triggered, is the time spent executing the different remediations. It is therefore recommended not to set too many remediations for the same alert, as the overhead may become too high.

Secondly, the space requirement in terms of storage. Considering the remediations are already stored somewhere, the overhead is minimal, as it only involves adding a configuration table.

Thirdly, the effective cost. There are no additional licensing costs for utilizing these SNow features, as a new contract has been recently secured that covers them.

Lastly, the impact on the ServiceNow platform was considered. Having a demanding application can overload the SNow instance and could impact the other tools that rely on it. With properly configured workflow triggers and application scope visibility, the impact is minimal. It is sufficient to configure a filter on the incoming alerts and trigger the workflow only for those relevant to the SRE team.

8.2 Implementation details

To be able to utilize REST APIs of technologies, such as AWX or Jenkins, that are in another network with respect to SNow, it is necessary to configure a path. Setting up a MID server is essential in order to do so, and additional internal network policies must be configured to enable the communication. Fortunately, this configuration has already been done for AWX and Jenkins.

8.2.1 Integration with AWX

The integration with AWX is already established, as it is actively utilized. An Ansible Spoke is available, providing various utilities for interaction.

When a new template needs to be triggered, a new record is created in a specific "Ansible table", storing the information about the playbook to be executed. A workflow then reads the newly created Ansible records and calls the REST API to trigger the playbook execution. The result of the REST API call is stored in a new record within the integration transaction table. Another workflow reads the integration transaction table and updates the Ansible record with the AWX Job ID. To obtain the execution output, the REST API must be called again using the Job ID, which can be done through another utility in the AWX Spoke.

Since this Spoke is defined in the global space, it can be utilized in our application without further modifications.

8.2.2 integration with Jenkins

Once the AWX integration is complete, the focus shifts to integrating Jenkins. Given that our team has numerous Jenkins jobs already established, the ability to call them directly would be advantageous. Otherwise, it would necessitate either rewriting the remediation actions in ansible playbooks, if feasible, or adding an intermediary step to call the Jenkins jobs from ansible playbooks.

The Jenkins integration is less developed compared to AWX, as most automation-triggering tools within our company primarily utilize AWX (such as ROSA or the Auto-remediation configuration).

Two main components are required to connect with Jenkins: the actual connection to jenkins' network and valid credentials. In order to set up the connection to the jenkins' network a connection record that specifies the url and the MID server to use has to be set up.

In the case of SRE's, the used pipeline - *pipeline2-ops.jenkins* - is in the global network. A couple of MID servers that connects to the gob network were found in the existing ones, grouped in a small cluster called "mid-int-gob-cluster". After creating the record it is possible to test it out by creating a dummy action that calls the jenkins' pipeline and checking the result. If the action returns a 401 Unauthenticated error (or anything but a Not Reachable error), it indicates that the connection is well configured but the credentials are not set up yet.

Regarding the credential, anything valid for that jenkins instance will suffice. Firstly a personal token was used for testing purposes (Figure 8.3), later the token for an azure robotic account was set up for real use scenario. Additionally a credential record must be created to configure the credentials in SNow.

To generate the personal token it is necessary to navigate to the personal profile, go to configuration, and "add a token". The object ID will be the username.

It is possible to test it using Postman and performing a POST request to the Jenkins job's URL (e.g., link of the job/build).

Since automation should not be triggered from a personal user account, a token was created from an AZ robotic account. The token generation is managed by another team

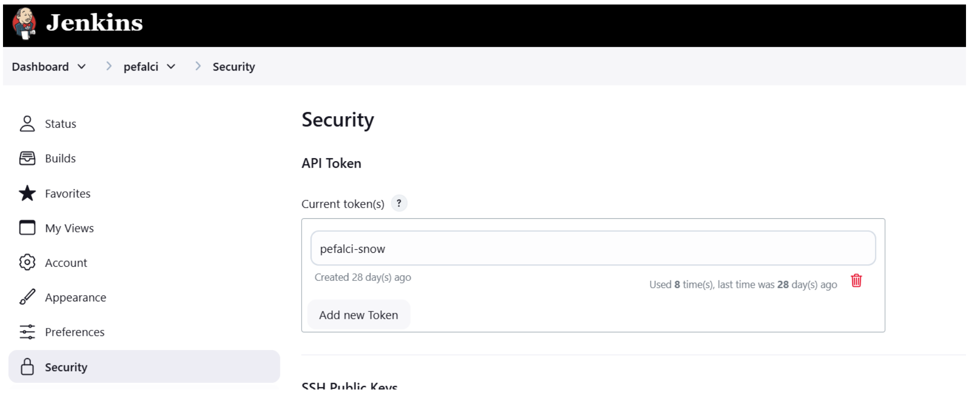


Figure 8.3: Jenkins Personal Token

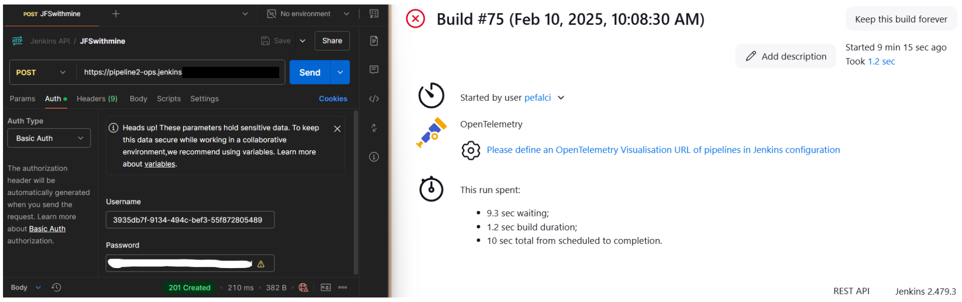


Figure 8.4: Jenkins Postman Personal Token

within the company, so it was enough to make a request. With the token, the credential record in SNow can be configured (the username is the object ID of the robotic account, and the password is the token).

Lastly the alias record must be created in order to select it easily from the actions. The three components: connection, credentials and alias are shown in Figure 8.5.

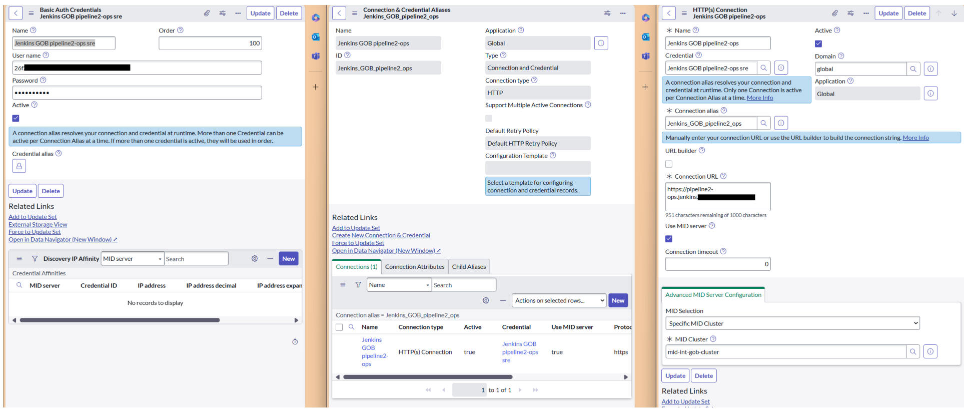


Figure 8.5: Connection, Alias and Credential

Once the connection is established, the next step involves implementing the necessary actions.

An action is required to initiate the execution of the Jenkins job, using configured connection and credentials to send a request to the Jenkins pipeline (Figure 8.6). Another action is needed to retrieve the output of the Jenkins job upon completion, calling the Jenkins REST API with the job ID to obtain the execution results.

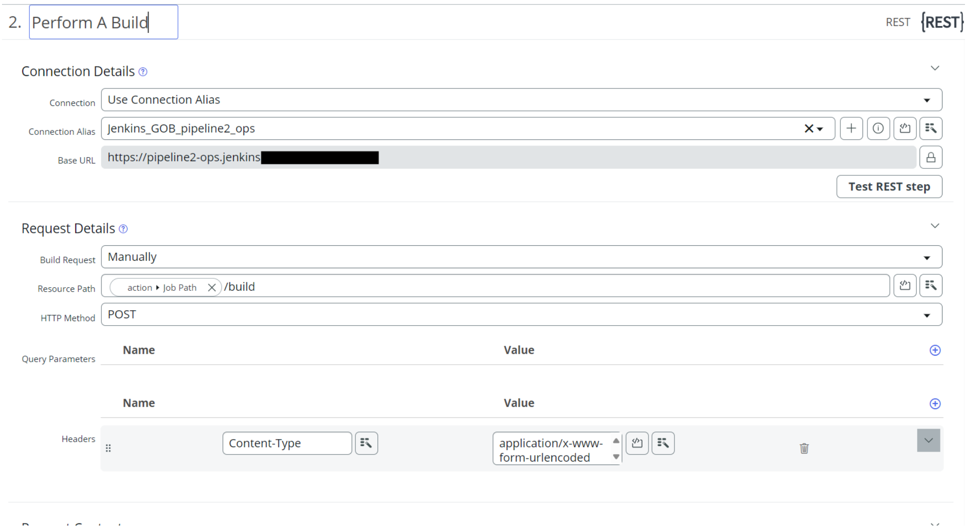


Figure 8.6: Jenkins Action

8.2.3 integration with MSTeams

By integrating with Microsoft Teams, real-time notifications can be sent to specific channels, keeping the team informed about significant events, such as the completion of Jenkins jobs or AWX tasks. Given that the tool is new, it is essential to monitor all activities. Sending notifications to Microsoft Teams creates a record of events and actions, which is beneficial for tracking purposes, as otherwise they could get lost in the sea of alerts.

In this scenario, a MID server is not required as the target is a Microsoft server. It is sufficient to set up a webhook in Microsoft Teams and make a REST API call from the SNow server. The webhook is configured in the desired channel for notifications. The card format is used to send these notifications[22].

To test the setup, a POST request can be sent to the webhook URL using Postman, similar to the process described in the previous chapter.

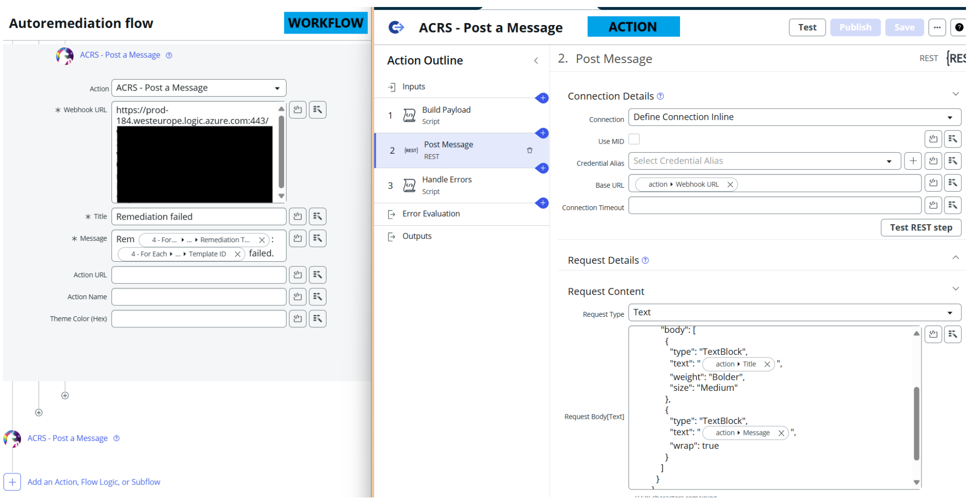


Figure 8.7: MS Teams Action

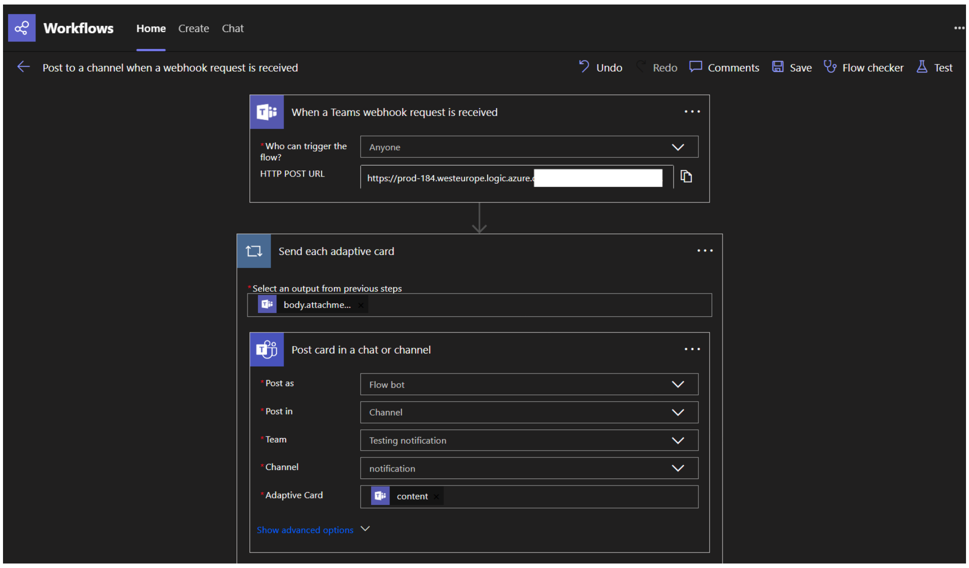


Figure 8.8: MS Teams Workflow

8.2.4 Splunk escalation

Several adjustments were made to support Splunk alerts.

The initial step involves setting up an alert in Splunk with an escalation to ServiceNow (SNow). This escalation is already feasible, and the fields are quite similar to those in ARGOS. However, there are additional considerations. Unlike ARGOS, Splunk did not have an escalation configured with a ”Clear” severity level. This configuration is necessary to close the SNow alert when the alert ceases, providing feedback on the remediation action. Unfortunately, this could not be implemented directly, as the escalation is managed by other teams who are currently working on it.

Once the escalation is configured, a new alert management rule was created in SNow to trigger the job when an alert from Splunk is generated.

Part IV

Conclusion

Chapter 9

Conclusion and future works

This thesis, conducted during an internship in the SRE HOS Amadeus team, investigated the implementation of self-healing mechanisms in cloud applications, with the objective of minimizing recovery time and avoiding manual intervention.

Initially, the study examined the built-in self-healing mechanisms in key infrastructure components, such as Couchbase and Kubernetes. These systems feature automated recovery capabilities, going from the ability to auto-failover to backup node, to detecting malfunctions of the system and restart it. The out-of-the-box functionalities do not address all failure scenarios, necessitating the use of external monitoring and automation solutions.

The thesis then examined the necessary integration of external monitoring and automation tools, such as ARGOS and Splunk for monitoring and Jenkins or AWX for remediations.

The ”Auto Remediation of Alerts” framework, developed by another team in AMADEUS, was identified as a first step towards a more autonomous incident resolution system. This system automates responses to recurring incidents by triggering predefined remediation actions, thereby reducing alert noise and enhancing operational efficiency. It was used to implement an auto-remediation able to automatically solve the couchbase failover node issue. The research demonstrated that automated remediation significantly reduces the mean time to recovery and decreases manual workload.

In order to tackle more complex use cases, an advanced application, specific for the SRE team was developed. This newly developed tool, still based on a rule-based system, integrates with more technologies and implements features such as a retry mechanism, the possibility to define multiple possible remediations. Most importantly, being owned by the SRE team, it can be easily maintained, extended, and modified in the future.

Although many foundational elements are now established at Amadeus, the journey towards a fully autonomous remediation system continues. Several areas remain open for further development. One of the key areas for future work is to expand the framework to cover additional use cases as they arise.

Current alerting mechanisms may generate redundant alerts, leading to unnecessary remediations. Implementing advanced correlation rules in ServiceNow would group related alerts and treat them as a unique issue.

Another area of improvement would be changing the decision system, currently based on a rule-based system. Future work could investigate the use of AIOps (Artificial Intelligence for IT Operations) to analyze historical incidents, identify patterns, and dynamically suggest and trigger remediation actions. AI technologies like AI-powered root cause analysis (AI-RCA) could be used to help identify the underlying causes of incidents. Implementing machine learning models to predict failures before they occur could further enhance system resilience.

By addressing these areas, future work can further improve the reliability and efficiency of cloud applications, ensuring that self-healing systems continue to evolve alongside modern infrastructure challenges.

Bibliography

[1] Amadeus. *Amadeus*. 2025. URL: https://amadeus.com/en.

[2] *Amadeus IT Group SA Number of Employees*. 2024. URL: https://www.macrotrends.net/stocks/charts/AMADY/amadeus-it-group-sa/number-of-employees.

[3] AWX. *Self-Healing Architecture*. 2025. URL: https://aws.amazon.com/solutions/guidance/self-healing-code-on-aws/.

[4] CNCF. *Decoding the Self-Healing Kubernetes, Step by Step*. 2020. URL: https://www.cncf.io/blog/2020/05/26/decoding-the-self-healing-kubernetes-step-by-step/.

[5] Couchbase. *Cluster Manager*. URL: https://docs.couchbase.com/server/current/learn/clusters-and-availability/cluster-manager.html.

[6] Couchbase. *Cross Data Center Replication (XDCR) Overview*. URL: https://docs.couchbase.com/server/current/learn/clusters-and-availability/xdcr-overview.html.

[7] Couchbase. *Intra-Cluster Replication*. URL: https://docs.couchbase.com/server/current/learn/clusters-and-availability/intra-cluster-replication.html.

[8] Couchbase. *XDCR Management Overview*. URL: https://docs.couchbase.com/server/current/manage/manage-xdcr/xdcr-management-overview.html.

[9] GeeksforGeeks. *What is Kubelet in Kubernetes?* URL: https://www.geeksforgeeks.org/what-is-kubelet-in-kubernetes/.

[10] Amadeus IT Group. *ACRS Service Level*. 2024. URL: https://AMADEUS.atlassian.net/wiki/spaces/HGM/pages/1293638759/ACRS+Service+Level?pageId=1293638759.

[11] Amadeus IT Group. *HOS High Level Design*. PowerPoint presentation. 2023.

[12] Amadeus IT Group. *Mandatory tags for CMDB population*. 2023. URL: https://AMADEUS.atlassian.net/wiki/spaces/SFCI/pages/1513390645.

[13] Amadeus IT Group. *ServiceNow Discovery*. 2024. URL: https://amadeus.atlassian.net/wiki/spaces/SFCI/pages/1513390397/ServiceNow+Discovery.

[14] Amadeus IT Group. *ServiceNow Incident Impact and Urgency Mapping Based on Event Severity*. 2023. URL: https://amadeus.atlassian.net/wiki/spaces/AISKB/pages/2224419346/ServiceNow+Incident+Impact+and+Urgency+Mapping+Based+on+Event+Severity.

[15] Amadeus IT Group. *Snow forwarder ServiceNow.* 2023. URL: https://amadeus.atlassian.net/wiki/spaces/MCONV/pages/2239012189/Component+-+Snow+forwarder+ServiceNow.

[16] Amadeus IT Group. *The Site Reliability Management Home.* URL: https://amadeus.atlassian.net/wiki/spaces/MFG/overview.

[17] Amadeus IT Group. *UHA Study.* 2025. URL: https://AMADEUS.atlassian.net/wiki/spaces/ASTSAR/pages/1005622999/UHA+study.

[18] Kubernetes. *Kubernetes Architecture.* URL: https://kubernetes.io/docs/concepts/architecture/.

[19] Kubernetes. *Liveness, Readiness and Startup Probes.* URL: https://kubernetes.io/docs/concepts/configuration/liveness-readiness-startup-probes/.

[20] Kubernetes. *Node Pressure Eviction: Self-Healing Behavior.* URL: https://kubernetes.io/docs/concepts/scheduling-eviction/node-pressure-eviction/#self-healing-behavior.

[21] Kubernetes. *Pod Lifecycle.* URL: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-restarts.

[22] Microsoft. *Create Incoming Webhooks with Workflows for Microsoft Teams.* URL: https://support.microsoft.com/en-us/office/create-incoming-webhooks-with-workflows-for-microsoft-teams-8ae491c7-0394-4861-ba59-055e33f75498#:~:text=Select%20More%20options%20next%20to%20the%20channel%20or,needs.%20Each%20template%20has%20a%20different%20authentication%20type..

[23] Microsoft. *Node Auto Repair.* URL: https://learn.microsoft.com/en-us/azure/aks/node-auto-repair.

[24] Microsoft. *Recommendations for self-healing and self-preservation.* 2023. URL: https://learn.microsoft.com/en-us/azure/well-architected/reliability/self-preservation.

[25] Microsoft. *Self-Healing Architecture.* URL: https://learn.microsoft.com/en-us/azure/architecture/guide/design-principles/self-healing.

[26] Niall Richard Murphy et al. *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media, 2016. URL: https://sre.google/sre-book/introduction/.

[27] RedHat. *How to Implement Self-Healing Infrastructure.* 2023. URL: https://www.redhat.com/en/blog/how-implement-self-healing-infrastructure#:~:text=Self-healing%20infrastructure%20leverages%20historical%20data-driven%20insights%20and%20automation,smart%20system%20management%20process%20provides%20more%20streamlined%20operations.

[28] ServiceNow. *MID Server.* 2021. URL: https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/success/quick-answer/mid-server-basics.pdf.

Notes

In order to write this thesis AI models have been used to:

- Correct grammar and syntax (Writefull)
- Re-elaborate notes with the prompt: "Make the paragraph impersonal and informal, with a style suitable for a thesis" (Copilot)

Aknowledgements

There are so many people that i met, interacted with, that I don't even know where to start. Each one of you has been a part of this journey, and I am grateful for that.

La prima che voglio ringraziare sei tu Sara, ti sei fatta carico di tutto il resto. Senza di te non avrei mai avuto la tranquillità di fare questo percorso.

Mamma. Gazie per avermi insegnato a essere resiliente e a non arrendersi.
Papà. Grazie per avermi sempre supportato e per credere in me.

Cecile et Philippe. Merci beaucoup pour m'avoir accueilli chez vous et pour m'avoir aidé à m'intégrer dans la vie française. Vous etes comme une deuxième famille.

Riccardo. Grazie per esserci sempre, anche in altre time zones. Dai pasti in mensa alle crisi sui programmi che devono eseguire.

Eleonora. Grazie per sopportare gli sbalzi d'umore, giuro che verró a trovarti a Volpiano, un giorno.

Laeticia. Merci per aver normalizzato gli abbracci giornalieri, per le gare di lasagne, per i piani di scale e per le chiamata update in francitaliano.

Alessandro, Giacomo e Simone. Gli amici di sempre, grazie per avermi dimostrato che le amicizie possono durare nel tempo. Grazie per i viaggi, per le chiacciere e per i momenti di relax.

Angela. Grazie per tutti i gelati.

Edoardo e Francesco. Grazie per avermi fatto sentire a casa, per i momenti di gioia (tra pizzette e feste) - e di sclero. Vi regalo un gelatino.

Mariia Ilyana e Andrey. Thank you for being my friends, for the time spent together, for the laughs and the memories.

Gatto. Grazie per avermi insegnato che c'é sempre tempo per dei gamberetti all'aglio, un buon film e un gioco da tavola.

Alessandro e Laura. Grazie per avermi insegnato a credere in me stesso, grazie per avermi fatto capire che la vita è fatta di scelte e che bisogna saperle fare.

Elena Alessio Marco Enrico e Filippo. Grazie per le stupende giornate di scacchi, gli infiniti eventi e la terribile amministrazione. É stato uno spasso affrontare tutto insieme.

Elie. You were always patient and you put trust in me. You teached me to respect the work-life balance. You encouraged me to investigate my ambitions and supported me in my choices. You are a great mentor and a great friend. Not thank you for winning the ping pong match though, now I have to go to RESA.

A final thanks to all the guys in the SRE and PMT team for being so welcoming and for the time spent together. Made this experience unforgettable.