Aalto University
School of Electrical
Engineering

Microservice Architectures and Serverless Computing COM-EV002

Final Design Report

Automatic Patch Management

Manish Kumar

MSc Cloud and Network Infrastructures
Aalto University, Espoo, FI

Table of Contents

1. Abstract

This design report aims to define the goals, environment, and architecture of an Automated Patching system that uses automation and a few other technologies to improve the experience of system patching, particularly for a cluster of servers.

The fundamental purpose of this system is to provide an efficient and accurate method of patching multiple servers simultaneously while eliminating the risk of downtime, BSOD (Blue Screen of Death), and, most significantly, human mistakes while commencing patches for production systems.

The system focuses on Windows server operating systems as well as Linux distributions such as Ubuntu and RHEL, assuming the workloads are hosted either in a public cloud environment or on-premise. Overall, the system will be based on a loosely coupled service-oriented architecture with explicit boundaries.

2. Overview

Security has always been a top priority and essential to every industry. Since more businesses are adopting serverless and cloud computing models, system patching is the most fundamental security and vulnerability mitigation layer. Apple and Microsoft, for example, have patch release periods for their respective operating systems.

Apple, for instance, releases critical patches for its hardware on the second Monday of each month, while Microsoft does the same on the second Tuesday. A zero-day vulnerability is an exception, a flaw for which no official patch or security patch has been provided. [1]

The Automated Patching System for Servers is intended to provide an automated patching solution to multiple servers or servers within a cluster. Without human intervention, this system will employ automation to push security and vulnerability patches. This approach's functional requirements include the ability to schedule patching activities on a specific day and during a certain downtime.

Next is Cluster Patching, in which the system should patch multiple servers in a cluster across multiple geographical regions simultaneously.

Finally, patch management, rollback options, patch approval, monitoring, report generation, and notification after the systems are up are additional requirements that ensure this solution is optimal and feasible.

2.1 Purpose

Companies can spend a hefty amount to remain compliant with their audits but fail to receive an optimal solution for system patching. Some solutions exist but make little sense regarding pricing and complexity, such as SCCM.

Moreover, out of numerous patches released on the second Tuesday, production servers only require one single security patch, which is very complex, time-consuming, error-prone and risky when done manually.

Therefore the motivation behind the implementation is to eradicate all such errors and risks and provide an optimal solution, eventually reducing the risk of security breaches and unexpected downtime.

2.2 Context of the Design

Assume a critical healthcare MNC has its entire workload housed on a public cloud, such as Microsoft Azure or AWS, with 400 servers in production. All 400 servers are internally connected via the same virtual network, and a few rely on failover clustering, which involves a group of separate and explicit servers working in sync to maintain high availability; workloads are exchanged when one server fails. [2]

These servers host apps, production websites, and customer health information vital to the organization's operations.

A system patching and vulnerability fix is required each month for a company like this to remain compliant with all the audits and be secure.

Although manual patching is still possible, consider the level of complexity and human error that could occur while performing the fixes, specifically when executing the failover manually.

As a result, the system intends to automate the patch management process for these servers, which includes installing security updates and addressing vulnerabilities regularly to guarantee that the servers are up to date.

Other aspects that may be considered in the context of this design and system are the number of servers, the complexity of "business architecture," and, most crucially, the technical expertise of the administrators.

Understanding the context of this system is therefore crucial for building a system that meets a company's specific needs and managing the patching process for its servers effectively.

2.3 Additional Context

To comprehend how the system operates from a top-view perspective, more context is required. To begin, because the servers are either on-premises or in the cloud, the administrator must install the "Agent" into the servers to enable complete system functionality to perform patching and other use cases (explained in section 3.2).

This agent is in charge of the servers reflecting on the system, allowing the administrator to determine when the server is ready for automated patching, which is basically, to show if the server is online.

We assume in the design that the agents are deployed on the servers and that the servers are reflected "online" in the automated patching system.

This will also ensure communication between the server and the system to monitor the backup status and real-time updates during patch deployment.

3. System Usage

3.1 Roles and Actors

The Automated Patching System primarily interacts with two groups: administrators with the patch management system, as well as servers with the patch management system reporting and notifying the status. The entire system will rely on the virtual system administrator (a server), that has the role of carrying out all the functional requirements.

Additionally, The patches and released vulnerability fixes are pulled from a centralized database (patch repository) so the administrator can review the list of released patches. [3]

The role of the administrator: The system is primarily intended for usage by administrators who manage and apply security patches within the cluster and to the servers.

Admins will interact with the patch management system via a web-based graphical user interface, allowing them to manage all patching activities, approve patches, and monitor and plan rollback along with all other miscellaneous activities.

In detail, their roles include:

- Implementing scripts for users to log off before downtime begins.
- Onboarding the servers to the system for patching and monitoring.
- Defining patching schedules based on the patch release cycle.
- Approval of specific patches before they are deployed to servers.
- Monitoring and generating the status of patching activities.
- Troubleshooting during the process.

Secondly, the entire system will contain one Virtual System Administrator (Typically a server) that takes control of all the functionalities and directly interacts with the end servers and the cluster to apply patches.

Finally, the Virtual server within the system will coordinate the patching activities and eventually generate a report with each server's patching status.

In detail, the roles include

- Automatic downloads from the centralized database [3] and applying patches to the servers.
- Reporting patching status to the patch management system along with notifying the admin.
- Providing logs on the failure of the patching activity.
- Roll back patches in case of issues and outages.

3.2 Use Cases

The use cases and additional context may be required to fully comprehend the Automated Patching System for servers. The following are the use cases:

1. **Pulling newly published patches from the repository:**
   The system will be able to pull newly released patches from the repository (Update catalog) so that the administrator can approve the patches for deployment before they are installed into the servers.

2. **Pre-checks:**
   The system will have the ability to perform certain pre checks before the patch cycle begins. The prechecks mainly check if the full backup is done in case of a rollback, and a snapshot of the operating system is taken before the deployment.

3. **Installation of patches:**
   To remediate vulnerabilities, the administrator can apply the critical patch to all servers in a cluster or a selected target—for example, the subscription.

   As a result, the system enables the administrator to schedule activity and simultaneously push patches to the servers.

4. **Monitoring and log generation of Patching Status:**

   The administrator will keep track of the patching status of all servers to verify that patches have been applied appropriately and that the boot sequence is followed.

   Once the patching activity is completed, the administrator will be able to generate a report with complete information on the patch, boot sequence, and known issues.

   Furthermore, the system will offer real-time status for monitoring and alert the administrator if any issues arise or when the patching is complete.

5. **Rollback:**

   This is a critical aspect of the system design; the system, as part of the prechecks, must do a server backup and take a snapshot before the deployment begins; hence, a rollback action plan is ready to be used in the event of issues and known boot loop issues.

4. System Architecture

Before the system design, it's essential to understand how the system interacts with each other and a few other crucial components.

Firstly, The web user interface and website that allows the administrator to manage the use cases are housed on a virtual server or virtual system administrator that is load balanced and configured for failover clustering. This virtual server is described in depth in 6.1. Furthermore, this virtual server is designed to maintain high availability consistently.

When the administrator is authorized via the SSO (Explained in 7.2) the API gateway works on routing the requests from the admin to explicit microservices, (7.3) and therefore, the admin can perform all the use cases and tasks associated with automated patching.

The patch management system, hosted on the virtual server, will house all the services, and each service will function as intended.

When the administrator is authorized via the API gateway, they can perform all the use cases and tasks associated with automated patching.

The patch management system, hosted on the virtual server, will house all the services, and each service will function as expected. The end servers are reflected online on the system with the help of an agent required for patching, reporting logs, and performing two-way communication with the services through a web socket, a communications protocol providing full-duplex communication via TCP. [4]

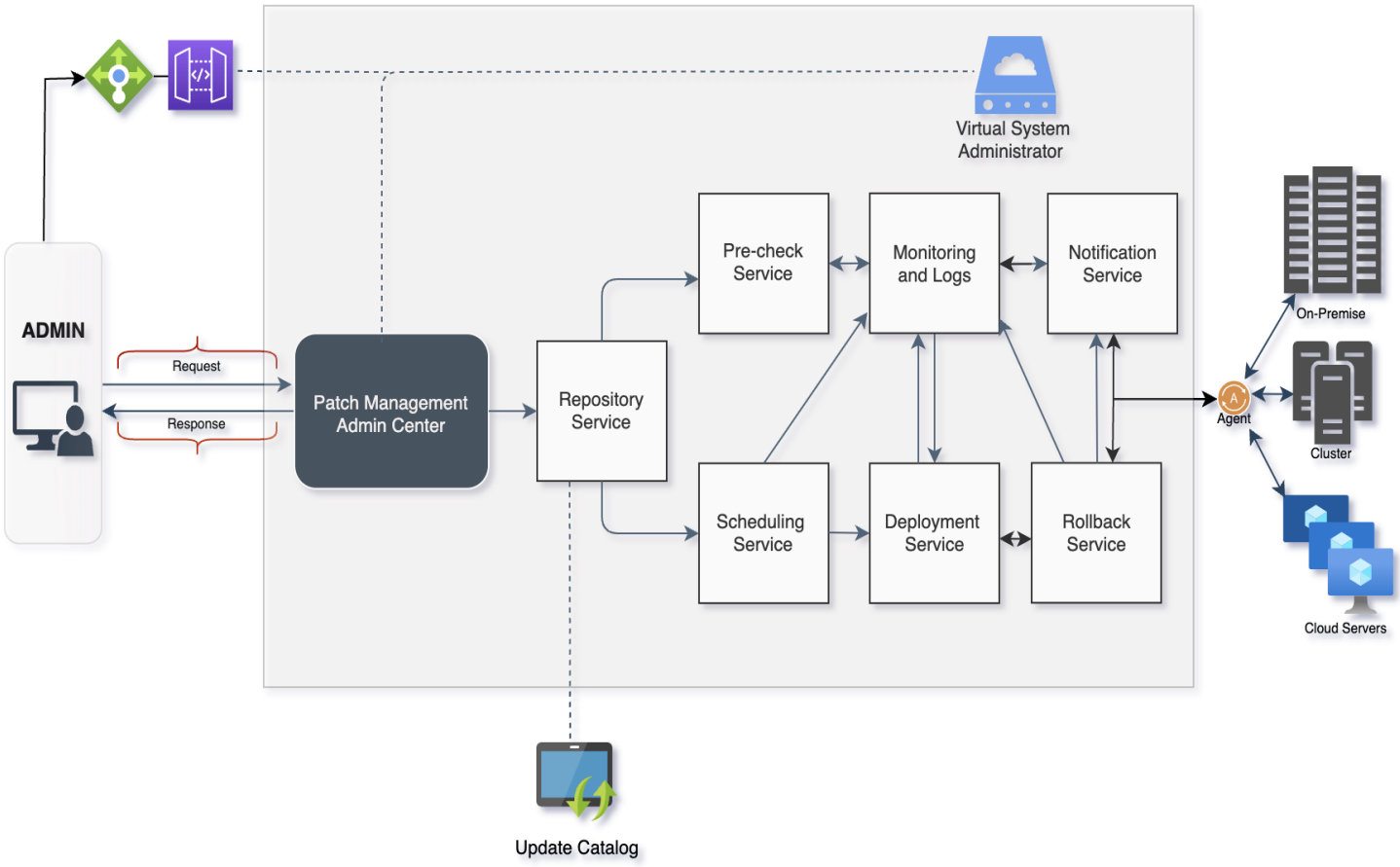

Figure 1. System Architecture for Automated Patching

5. Interfaces

Interfaces are essential for distinguishing the boundaries between each microservice so that they can function explicitly for specific use cases. Microservices are intended to be loosely connected. However, for the numerous independent microservices that make up the Automated Patching system architecture to remain modular and perform independently, they must be separated by boundaries.

It is crucial to comprehend the automated patching process to link how microservices could depend on other entities to accomplish their final state (or) achieve the last use case of rollback. Each microservice will function as an explicit process in its state, maintaining its persistence because all microservices are hosted via a virtual system administrator (a virtual server).

1. **Repository Service:**

This is the backbone of the automated patching system that will directly communicate with the WSUS server or the updated catalog from various vendors. The administrator first sends a request to pull a specific update using the patch management system. This service now has its interface connected to two primary services, the pre-check and the scheduling service, and proceeds to work internally to fetch and fulfill the request.

2. **Pre-check Service and Scheduling Service:**

The administrator then has the option to pre-check the end server and schedule the patches to apply at a specified set time and date when requests are pulled from the repository to push a specific set of updates to the end server. Each service operates autonomously, and an internal quartz scheduler and database are used.

The scheduling service has interfaces to the monitoring service and the deployment service, but the pre-check service has its interface and a connection from the repository and extends to monitoring.

3. **Monitoring and Deployment Services:**

When the scheduler plans the maintenance for a particular day and time, the first log is created when pre-checks are made with the end server. Monitoring extends its interfaces with the notification service, and the deployment service now includes interfaces to the rollback, monitoring, and notification services.

There are two separate causes for this. The deployment can first be stopped at any point by the administrator, who will then be notified by email, and the changes will be undone. Second, the administrator must be notified when the service reaches its end state (achieving the particular use case).

4. **Notification and Rollback:**

These microservices typically have a boundary and an interface that connect to the external environment through an "Agent" that further executes and delivers the changes while adhering to the use cases specified.

As a result, each service shares its interface with the agent, which handles two-way WebSocket communication and can reverse modifications in the event of a mishap and alert the notification service in case of success or failure.

If there is a pattern in the interfaces, it is essential to note that each service has its own set of boundaries and must be persistent and maintain its state. Secondly, even though each microservice operates independently, communication between them is necessary for the automated patching system to complete its task, which is to patch the cluster and servers and allow for rollback.

6. Service Architecture

6.1 Virtual System Administrator

This is not a microservice but rather an essential component of the system design that houses all of the services as well as the administrator's web-based user interface (i.e.) Patch management system.

The services mentioned (Microservices) in the system design will run as an explicit and a separate process in the virtual machine As a result, before introducing explicit services, it is critical to understand the architecture of the virtual server.

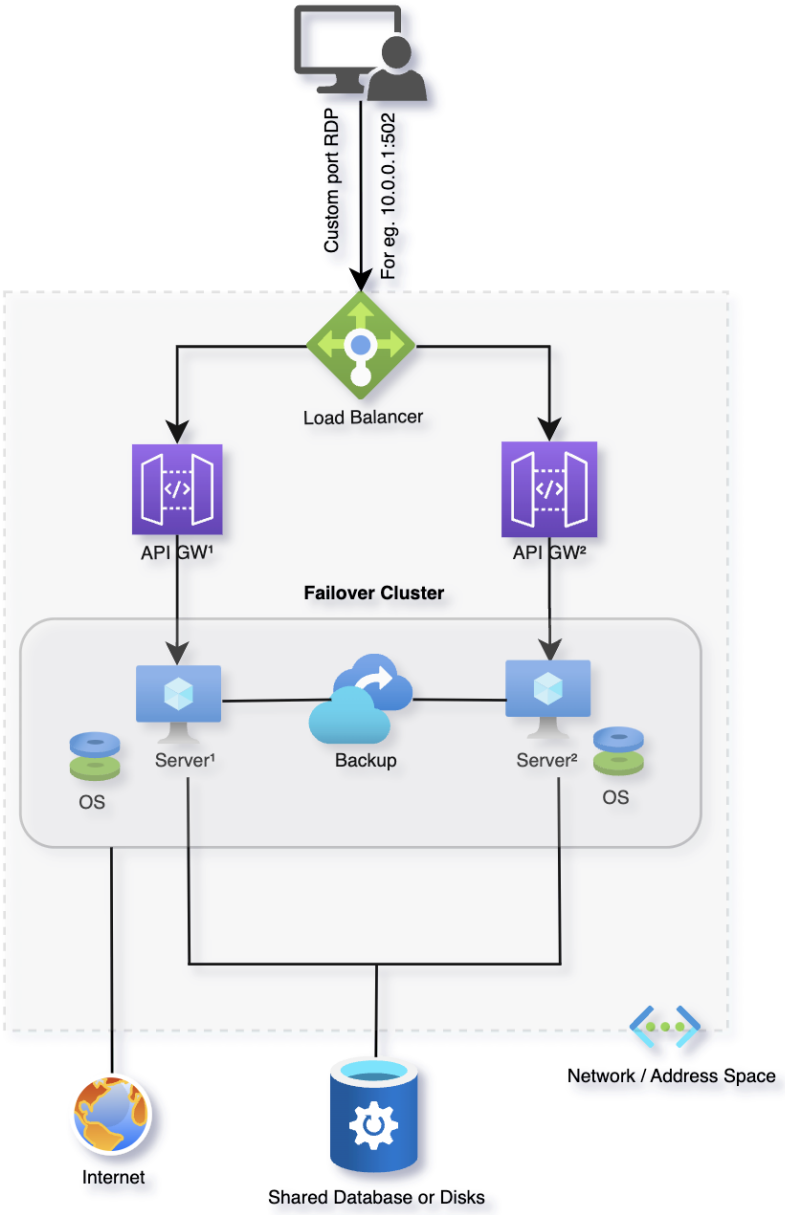

Figure 2. Architecture diagram showing the failover capability of the virtual System Administrator (Virtual Server)

With the help of this server, the patch management console can communicate with the internet and also with the Microsoft Catalog. [3]

Additionally, these servers are designed to be in a failover configuration, so there is always high availability whenever a server goes down, and has a shared database to process the use-cases. Lastly, the servers together are always fully backed up for an extra layer of retaining their state and persistence.

6.2 Repository Service

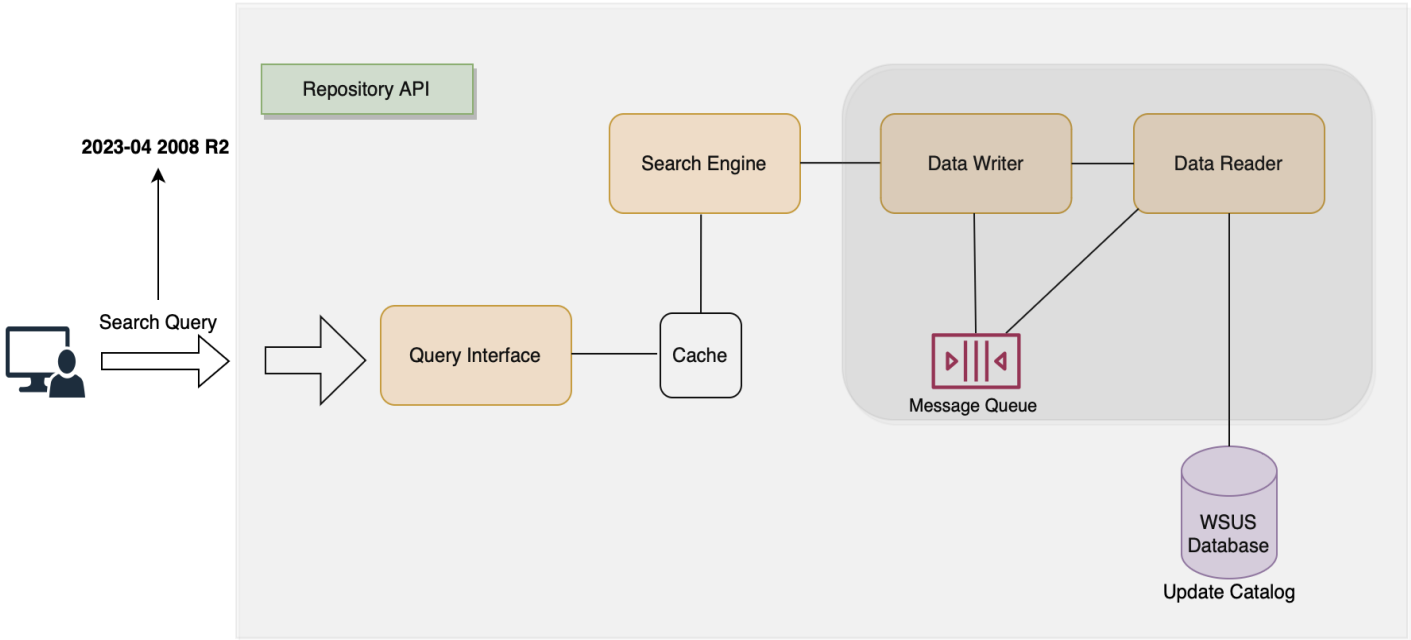

Figure 3. Service diagram of Repository API that communicates with the WSUS database to achieve the use-case of the failover capability of pulling new patches. For example, the query is “2003-04 2008 R2”

6.3 Pre-Check Service

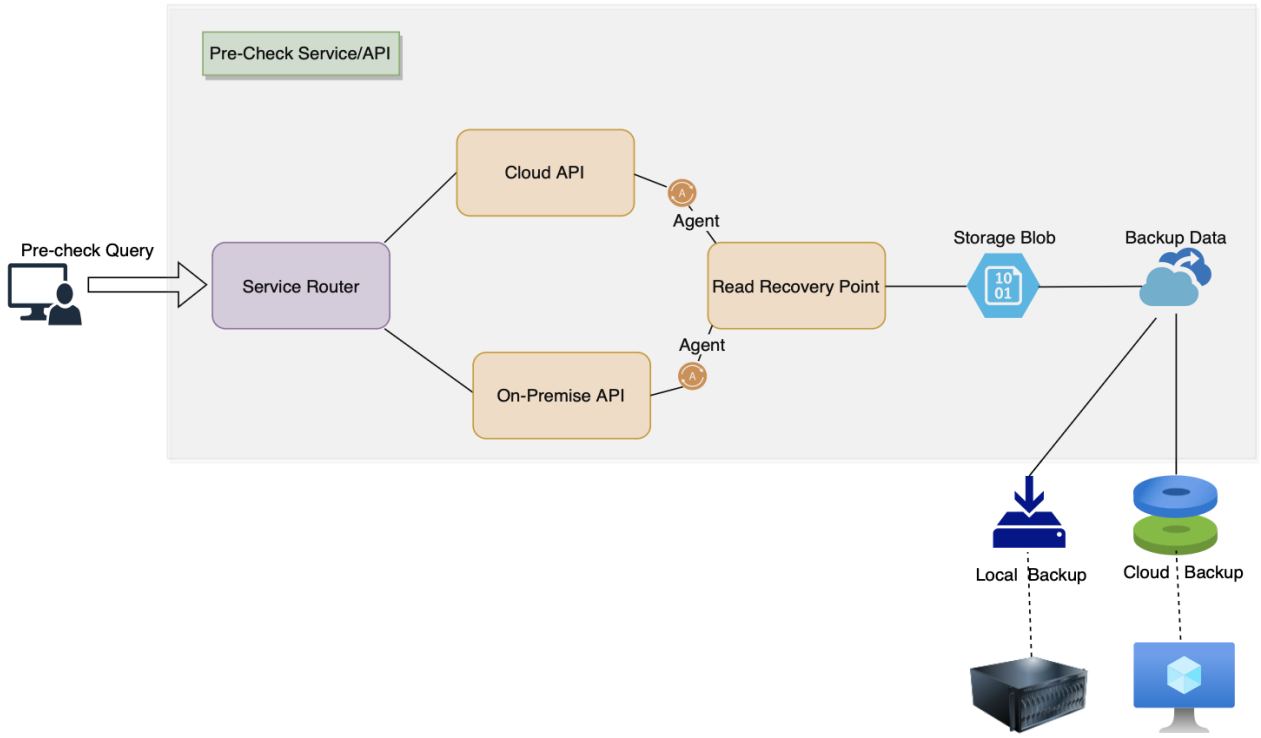

Figure 4. Service diagram of the pre-check API with a service router to route requests.

6.4 Scheduling Service

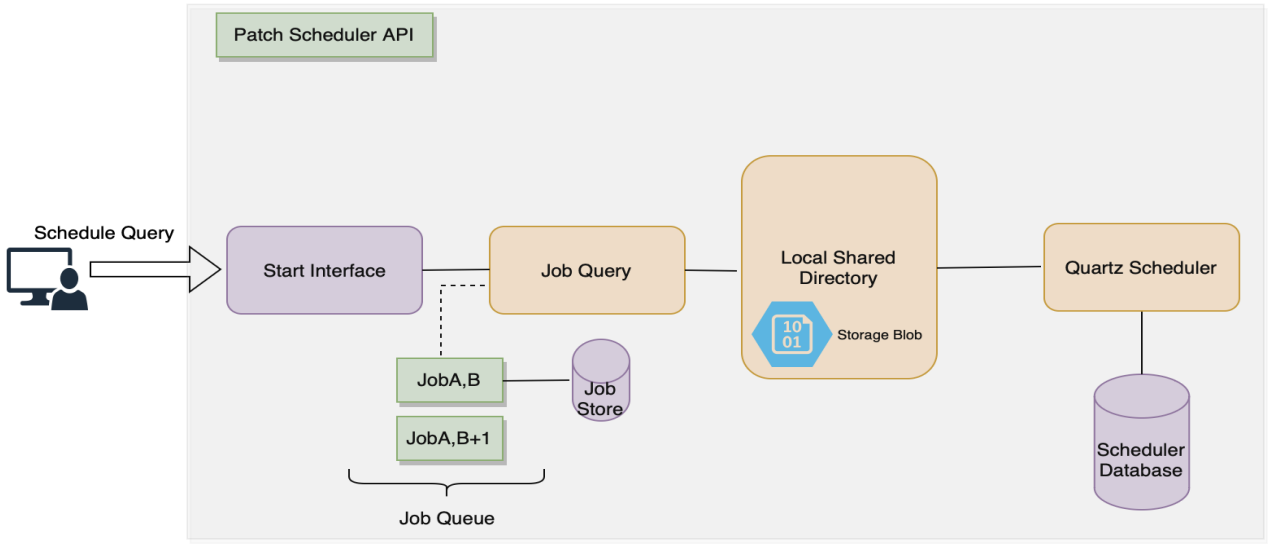

Figure 5. Service diagram of Scheduling Service API with quartz scheduler.

6.5 Monitoring Service

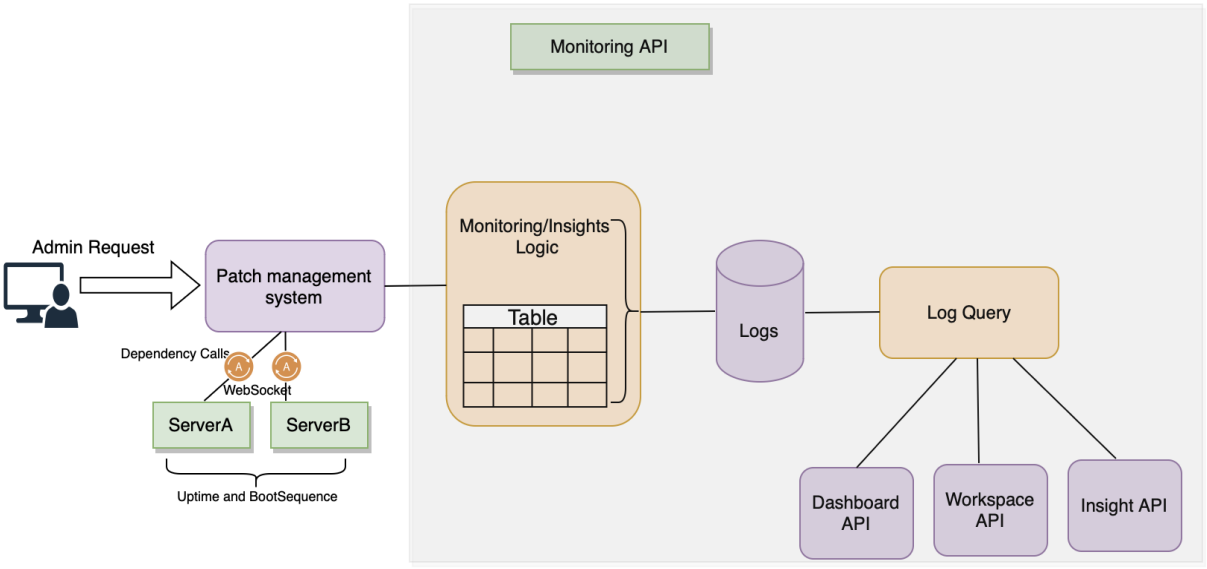

Figure 6. Service diagram of Monitoring Service with multiple APIs for monitoring.

6.6 Deployment Service

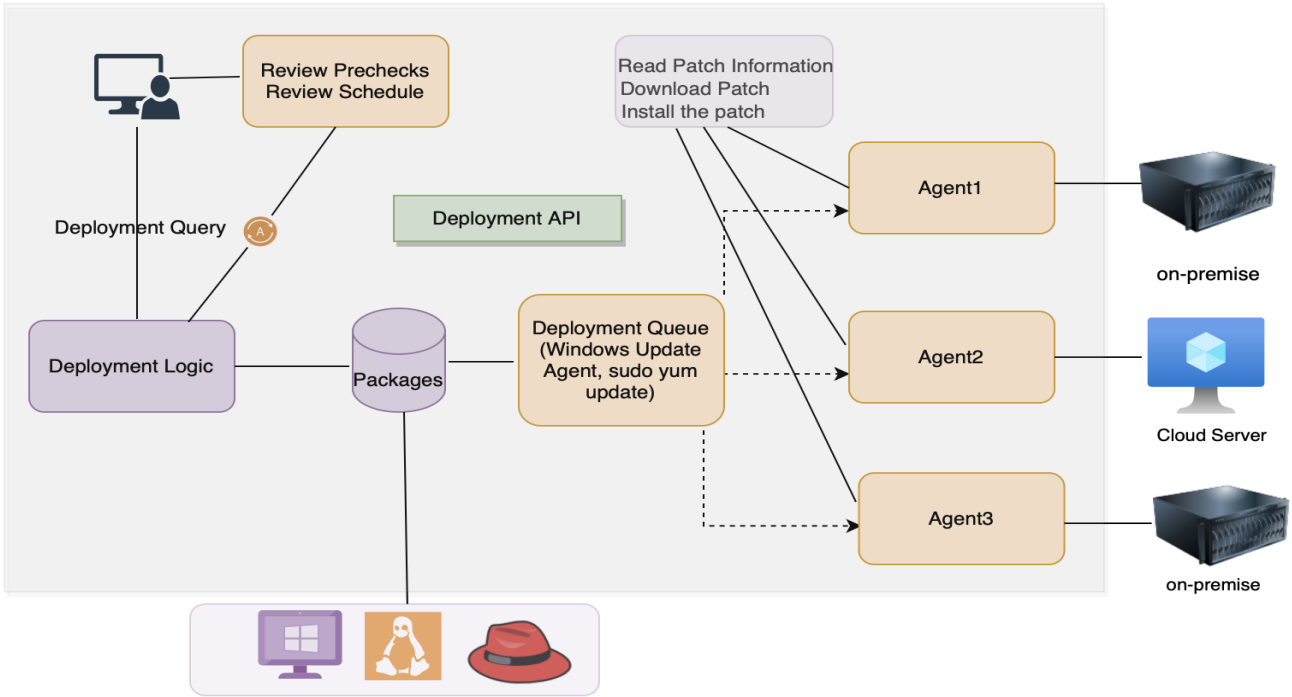

Figure 7. Service diagram of deployment service that communicates with external entities.

6.7 Notification Service

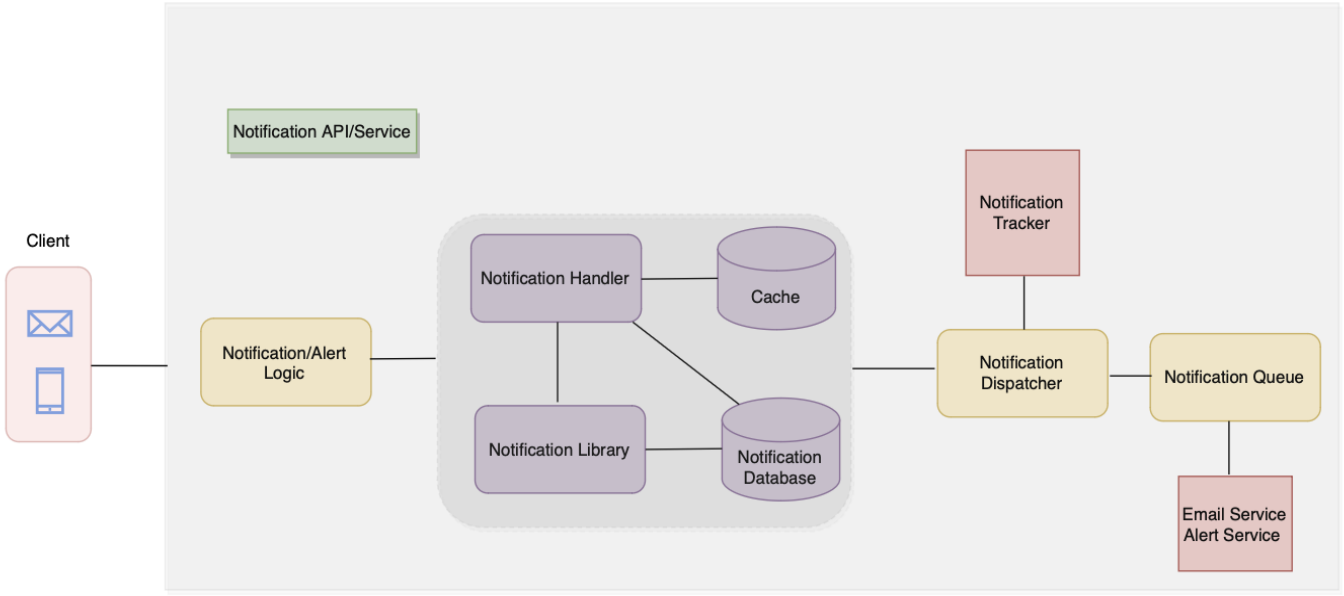

Figure 8. Service diagram of Notification and alerting service.

6.8 Roll-Back Service

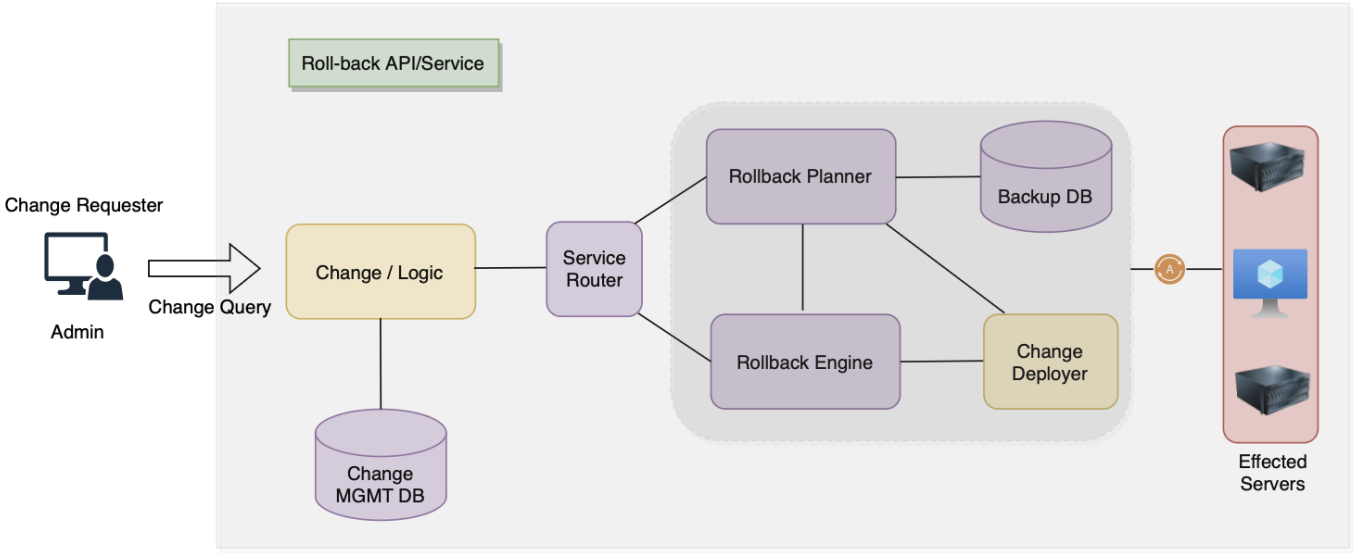

Figure 9. Service diagram of roll-back service essential for reverting changes.The service has an interface with the backup database to recover and swap the server corrupted disk.

7. Additional Critical Components

7.1 Communication of the agent from the end server to the virtual server

The agent is crucial in establishing contact with the end server and the patch management system in this system. It is critical to show that the servers are onboarded and ready to be patched, that logs and errors are being transferred in real-time from the end server, and that the server has followed a startup sequence after the patch has been deployed.

As a result, this agent employs a WebSocket, a communication protocol that enables real-time and low-latency full-duplex communication over just one link. This is highly popular in online gaming and chatbots because it remains open and allows data to flow between two endpoints for real-time web applications.

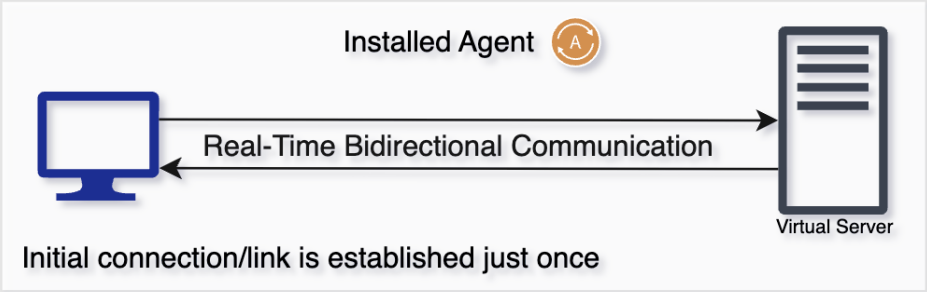

Figure 10. WebSocket operation in the agent

7.2 Mode of authentication for the administrator

The administrator must be authorized to gain admin-level access to the web interface, the automated patch management system. Because the design involves a cluster of production servers for maintenance, security must be applied at this stage for authentication. As a result, a single sign-on can be used, eliminating the need for the administrator to obtain authorization for every other internal service.

While the specifics of active directory and user-level permissions are not covered in this report, it is strongly advised that a comprehensive security design or a service must be established.

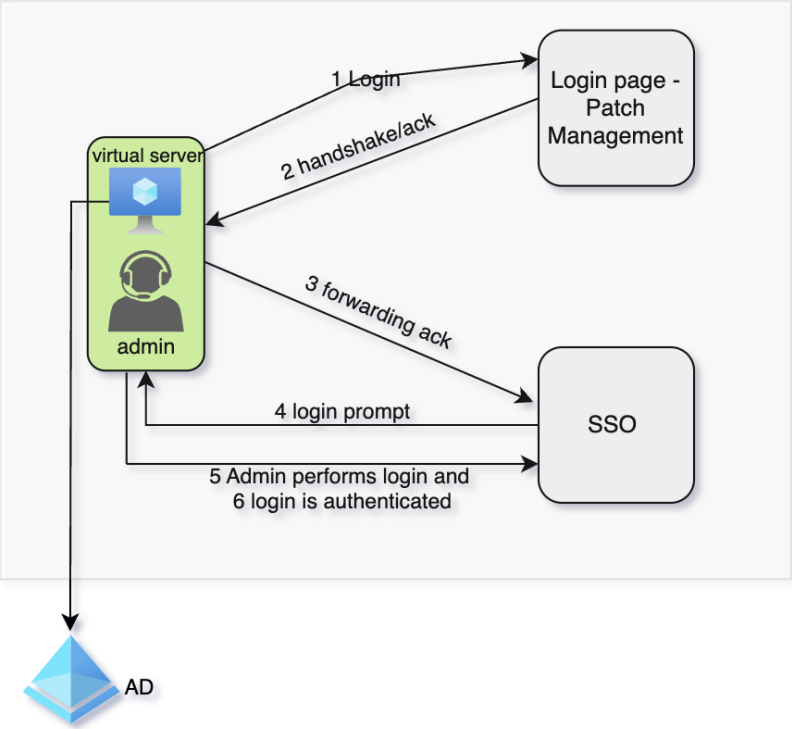

Figure 11. Authentication using an explicit SSO service

7.3 Role of API Gateway

The API Gateway could be a significant component in the design of an automated patching system for handling communication between different microservices -

The primary use of API GW is Service Discovery, and a reverse proxy which works on dynamically discovering and routing requests to explicit services and performs handling of all incoming admin requests and routing them to the necessary microservices for further processing. This often allows for a single entry point for all queries.

This significantly streamlines communication. Although API GW provides authentication and authorization, SSO has been implemented as a separate entity for this design.

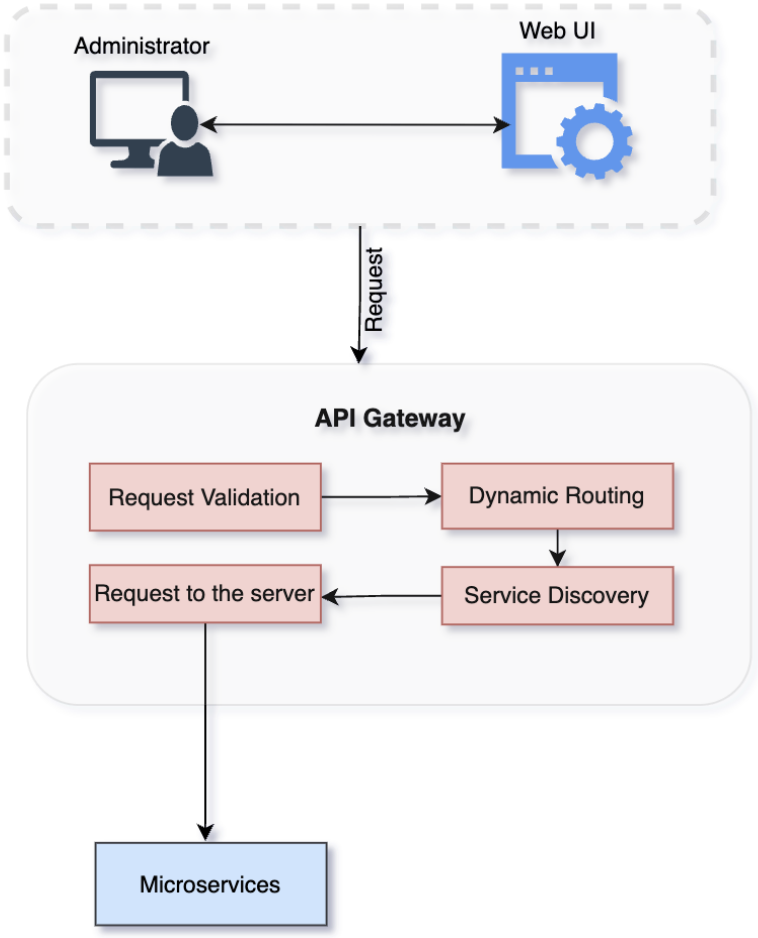

Figure 12. Implementation of API Gateway supporting the Virtual Server

7.4 Virtual Server - on-premise or cloud? Which is better?

Finally, when housing the virtual System Administrator (the virtual server), the ideal option always depends on several criteria for a long-term solution. For example, the server can be implemented on-premises or on the public cloud.Assume the solution has been running for five years.

According to Cameron Fisher's extensive study and research on Cloud vs On-Premise Computing, the cloud is more expensive in the long run. [5]

Cloud computing is a fantastic developing technology with significant public releases of new features almost daily, Microsoft Azure updates, for example. [6] This strategy typically entails paying for resources (Compute, Networking, Storage, and so on) on a subscription basis.

All maintenance, including patching, SLA maintenance, and other high availability issues, is handled by public cloud vendors. When it comes to cost savings, this model is unrivaled.

For example, a B series processor can be reserved for three years at a substantially reduced monthly fee. [7] Other resource categories, such as databases and disks, are also subject to reservations. This is highly cost-effective for small and medium-sized organizations, particularly for this solution that necessitates failover clusters and load-balanced servers.

As a result, additional support or personnel are not required to maintain the hardware in a cloud model. Most importantly, vertical scaling is as simple as clicking a button in cloud computing.

If the server is hosted on-premises, this will entail purchasing and maintaining the hardware and software infrastructure on-site and an explicit license for the operating system. This can be costly upfront but may be more cost-effective in the long run, especially for larger firms with consistent workloads.

As a result, hosting this automated patching system with reservations is an excellent approach to scrimp pricing and physical maintenance while maintaining high availability.

7.5 Is there a need for Caching in this model?

The service diagrams for each microservice in this system directly highlight the crucial importance that caching plays in this system design. Even though the servers are load-balanced, using caching lowers the burden on the server and ultimately prevents the failover scenario.

Most importantly, if the types of caching in the system design need more clarification -

1. **Content caching:** This entails caching resources like patches and vulnerabilities that are regularly accessed. The amount of moderated data on the server and the internet is eventually reduced. This is most helpful in distributed systems where different clients can access the same data. (Multiple servers in this instance)

2. **Result Cache:** This is important for the automated patching system since it caches queries and results, and if the administrator makes a similar query again, faster processing is anticipated. This eventually results in a lighter load on the database.

3. **Metadata caching:** Finally, retrieving system metadata, like the patching status, is part of metadata caching. This is important since there is no need to query the database, performance is greatly improved, and the database is not under as much stress.
   When it comes to abstract caching, synchronous and synchronous distributed caching, Abstract caching may not be required for this system because automated patching only needs to communicate with the vendors for released patches, which doesn't require retrieving enormous amounts of data much more frequently. To achieve high data consistency, distributed synchronous uses data replication across numerous nodes. Since the data being cached is typically not time-sensitive, this is optional in this design. However, distributed asynchronous has a time delay in sync but otherwise operates exactly like synchronous. Since it lessens frequent database interactions, this can be helpful in automated patching to increase performance and scalability.

8. References

1. Siosulli. (n.d.). Mitigate Zero-day vulnerabilities. https://learn.microsoft.com/en-us/microsoft-365/security/defender-vulnerability-management/tvm-zero-day-vulnerabilities?view=o365-worldwide

2. Awati, R. and Rubenstein, B. (2023) What is a failover cluster? – TechTarget definition, SearchWindowsServer. TechTarget. https://www.techtarget.com/searchwindowsserver/definition/failover-cluster

3. Microsoft Update Catalog. (n.d.). Microsoft Update Catalog. https://www.catalog.update.microsoft.com/Home.aspx

4. What is WebSocket and How It Works? (2021, October 28). What Is WebSocket and How It Works? https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is

5. Fisher, Cameron. (2018). Cloud versus On-Premise Computing. American Journal of Industrial and Business Management. 08. 1991-2006. 10.4236/ajibm.2018.89133.

6. Azure updates | Microsoft Azure

7. Pricing Calculator | Microsoft Azure. (n.d.). Pricing Calculator | Microsoft Azure. https://azure.microsoft.com/en-us/pricing/calculator/