|  |
| --- |
| **ADVANCED SOFTWARE ARCHITECTURES** **SOEN 6471**  **SUMMER 2023**  Deliverable 2  NGINX  Declaration  We, the members of the team, have read and understood the Fairness Protocol and the Communal Work Protocol, and agree to abide by the policies therein, without any exception, under any circumstances, whatsoever.  Team L  Abhimanyu Sharma - 40188795  Kirthana Senguttuvan - 40189746  Saurabh Sharma - 40226298  Urvish Rupeshbhai Tanti - 40197508  Vaibhav Sharma - 40197697 |

Table of Contents

[Problem 4: Quality Attributes 2](#_Toc138087452)

[Quality Attributes that NGINX aims to satisfy 2](#_Toc138087453)

[1. Efficiency 2](#_Toc138087454)

[2. Scalability 2](#_Toc138087455)

[3. Reliability 2](#_Toc138087456)

[4. Security 2](#_Toc138087457)

[5. Flexibility 2](#_Toc138087458)

[6. High Availability 2](#_Toc138087459)

[7. Deployability 3](#_Toc138087460)

[Quality attributes that NGINX should satisfy but does not 3](#_Toc138087461)

[1. Simplicity 3](#_Toc138087462)

[2. Usability 3](#_Toc138087463)

[3. Compatibility 3](#_Toc138087464)

[4. Configurability 3](#_Toc138087465)

[5. Interoperability 3](#_Toc138087466)

[6. Monitorability 4](#_Toc138087467)

[Problem 5: Orthogonal Views of NGINX 4](#_Toc138087468)

[Agile Modeling Practices used for the Views 4](#_Toc138087469)

[Agile Modeling Principles used for the Views 4](#_Toc138087470)

[Structural View 4](#_Toc138087471)

[Behavioral View 5](#_Toc138087472)

[Interaction View 5](#_Toc138087473)

[Problem 6: Architectural Patterns, Principles and Styles of NGINX 6](#_Toc138087474)

[a) Patterns, principles, styles, and/or tactics of software architecture used by NGINX 6](#_Toc138087475)

[Description of NGINX architecture 7](#_Toc138087476)

[b) Undesirables of NGINX 8](#_Toc138087477)

[ Inconsistent cache validation: 8](#_Toc138087478)

[ Inconsistent variable configuration: 8](#_Toc138087479)

[ Subrequest Limitations 8](#_Toc138087480)

[ Insufficient test coverage: 8](#_Toc138087481)

[ Incomplete log Management: 8](#_Toc138087482)

[Problem 7: Suggestions for improving NGINX architecture and architecture description 9](#_Toc138087483)

[Suggestions for improving Software Architecture Description of NGINX 9](#_Toc138087484)

[Suggestions for Improving Software Architecture of NGINX 13](#_Toc138087485)

[Performance Optimization 13](#_Toc138087486)

[Security Enhancements 13](#_Toc138087487)

[Monitoring and Logging 13](#_Toc138087488)

[Team Contributions 14](#_Toc138087489)

[References 15](#_Toc138087490)

# **Problem 4: Quality Attributes**

## Quality Attributes that NGINX aims to satisfy

### Efficiency

**Definition**: Efficiency of a software system can be defined as the resources expended in relation to the accuracy and completeness with which users achieve specified goals [1].

**Comments**: NGINX is designed to handle a large number of concurrent connections efficiently and with low memory usage. It employs an event-driven, asynchronous architecture to achieve high performance and minimize resource consumption [5]. NGINX includes caching mechanisms that can improve performance and reduce the load on backend servers. It can cache static content and even cache dynamic content, such as responses from backend servers, to serve subsequent requests more quickly [2].

### Scalability

**Definition**: The property of a system to handle an increased (or decreased) workload by increasing (or decreasing) the cost of the system is called Scalability [1].

**Comments**: NGINX is built to scale horizontally, allowing it to handle increasing traffic loads by adding more servers in a load-balanced configuration. It can distribute client requests across multiple backend servers, improving overall system scalability [2].

### Reliability

**Definition**: Reliability refers to the likelihood that a software product will operate without failure within a defined timeframe and under specific conditions. This likelihood is influenced by various factors, including the inputs provided to the product, the way it is used, and the presence of any potential faults. The inputs determine whether any existing faults, if present, will be encountered during the product's usage [1].

**Comments**: NGINX is known for its stability and robustness. It is designed to handle heavy workloads and high traffic volumes without compromising its performance or causing system failures. NGINX can also recover gracefully from failures, automatically restarting or isolating failed components [4].

### Security

**Definition**: The extent to which a software product safeguards information and data, ensuring that individuals, other products, or systems are granted appropriate levels of data access based on their types and authorized permissions [1].

**Comments**: NGINX provides various security features to protect web applications. It supports SSL/TLS encryption, allowing secure communication over HTTPS. NGINX can also act as a reverse proxy, providing an additional layer of security by shielding backend servers from direct exposure to the internet [4].

### Flexibility

**Definition**: Flexibility, as a quality attribute for software, refers to the ability of a software system to adapt and accommodate changes or variations in its requirements, configuration, and functionality. A flexible software system can be easily modified, extended, or customized to meet evolving needs and support different use cases.

**Comments**: NGINX offers a high degree of flexibility in configuration and customization. It provides a powerful configuration language that enables users to define complex routing rules, load balancing algorithms, and other application-specific settings. NGINX also supports dynamic module loading, allowing users to extend its functionality with custom modules [4].

### High Availability

**Definition**: Availability of a software can be defined as the degree to which a software product is operational and accessible when required for use [1].

**Comments**: NGINX supports high availability setups by providing features such as load balancing, failover, and active-passive configurations. These features ensure that web applications remain accessible even in the event of server failures or maintenance activities [4].

### Deployability

**Definition**: Deployability, as a quality attribute for software, refers to the ease and effectiveness with which a software system can be deployed to its target environment. It encompasses the process of preparing, installing, configuring, and launching the software in a production or operational environment. Deployability aims to streamline the deployment process, reducing effort, time, and potential errors.

**Comments**: NGINX is relatively easy to deploy and configure. It has a straightforward installation process and provides clear documentation and community support. NGINX also integrates well with various operating systems, containerization platforms, and orchestration tools, making it suitable for different deployment scenarios [4].

## Quality attributes that NGINX should satisfy but does not

While NGINX is a powerful web server and reverse proxy server software that excels in many areas, there are a few quality attributes that it does not fully satisfy. Here is a list of some quality attributes that NGINX does not completely address:

### Simplicity

**Definition**: Software Simplicity can be defined as the degree to which a software product or its component has a design and implementation that is easy to understand [1].

**Comments**: NGINX, with its powerful configuration language, has a steeper learning curve for beginners who are new to web server administration or have limited experience with command-line interfaces. It might require some time and effort to become proficient in configuring and managing NGINX effectively [5].

1. Usability

**Definition**: Usability of a software is the degree to which it can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction, in a specified context of use [1].

**Comments**: NGINX primarily relies on command-line configuration and administration, lacking a comprehensive graphical user interface. While third-party tools exist to provide a GUI for managing NGINX, the software itself does not offer a built-in graphical interface, which could be more intuitive for certain users [5].

### Compatibility

**Definition**: Compatibility of a software is the degree to which it can exchange information with other software products, and/or perform its required functions, while sharing the same hardware or software environment [1].

**Comments**: While NGINX can be installed and run on Windows systems, it was originally developed for Unix-like operating systems and does not have the same level of native integration and support on Windows. This leads to slight differences in behavior or limitations compared to its Unix counterparts [5].

### Configurability

**Definition**: Configurability refers to the ability of a software system to be easily customized and adjusted to meet specific requirements, preferences, or environmental conditions without modifying the underlying code.

**Comments**: NGINX requires manual configuration for various settings and does not offer built-in mechanisms for automatic configuration discovery or self-configuration. While NGINX's configuration language is powerful, it requires knowledge and expertise to set up and modify configurations appropriately. Also, although NGINX supports popular load balancing methods like round-robin and IP hash, more advanced algorithms require additional configuration or custom modules [3].

### Interoperability

**Definition**: Interoperability of a software is the degree to which two or more software products can exchange information and use the information that has been exchanged [1].

**Comments**: NGINX primarily focuses on serving static and dynamic content and acting as a reverse proxy. It does not provide built-in capabilities for content management systems (CMS) like WordPress, Drupal, or Joomla. Additional configurations or plugins are required to integrate NGINX with these CMS platforms effectively [4].

### Monitorability

**Definition**: Monitorability of a software refers to the degree to which it can be effectively monitored and observed to gain insights into its performance, behavior, and health. It encompasses the ability to collect, analyze, and visualize relevant data and metrics about the system's operations, resource utilization, errors, and other relevant aspects.

**Comments**: NGINX provides basic logging and monitoring features, but it lacks comprehensive real-time monitoring and analytics capabilities. Advanced monitoring and analytics tools are needed to be integrated separately to gain deeper insights into NGINX's performance and traffic patterns [3].

# **Problem 5: Orthogonal Views of NGINX**

## Agile Modeling Practices used for the Views

* Model with others: This modeling activity was a group activity with each group member actively contributing to the final increment of the model.
* Iterate to Artifact: Every team member was assigned a separate use case of NGINX to model over a UML diagram with different use cases serving as different artifacts to the three views.
* Model in Small Increments: Every team member’s use case was incrementally integrated into the use case.

## Agile Modeling Principles used for the Views

* Multiple Models: Currently showcasing 3 models and all of which have been developed post iterative feedback from the team members.
* Rapid Feedback: Indulged in multiple peer reviews for delivery of models and content thereof.
* Assume Simplicity: These models envision the most basic use cases while also envisioning over 90% of the use cases in which NGINX is used hence these assume simplicity.
* Incremental Change: As already mentioned, based on team member feedback, these models were changed incrementally.

## Structural View

A picture containing text, parallel, diagram, number

Description automatically generated

Figure: Highlights the interaction between components for NginX [6]

Network software called NGINX provides dependable and quick connectivity for contemporary applications. It has a control plane and a data plane as part of its architecture. While the data plane manages traffic routing and encryption, the control plane employs controllers to maintain desirable states across managed applications. NGINX Service Mesh employs Dynamic Admission Control through injection and adheres to the sidecar pattern. By including an init container and a sidecar, this modifies Pod setups. Based on Kubernetes events, the control plane creates an internal configuration and transmits it to application sidecars. Traffic redirection and configuration are handled by the sidecars, which are made up of an init container and NGINX Plus. The "brain" of the data plane is NGINX Plus, which controls mTLS, traffic routing, tracing, and monitoring. It follows traffic regulations set forth in Kubernetes resources and encrypts connections. The NGINX Plus sidecar, which proxies the traffic to its original destination, receives the traffic after it has been rerouted. The sidecar uses the Linux socket API to forward the communication while carrying out operations like encryption and decryption. Overall, NGINX Service Mesh uses controllers, sidecars, and NGINX Plus to divide networking and application concerns. For dispersed applications, this design offers dependable and secure network connectivity.

## Behavioral View

A picture containing text, diagram, parallel, line

Description automatically generated

Figure: Highlights how components handle client HTTPs request from initiation to delivery [7]

The Behavioral view examines the use of FrontEnd and BackEnd Proxies with their correlation to handle HTTPs requests. We can also observe the presence of Authentication Server to authenticate HTTPS requests from client end. HPC Node is used by both the Frontend and the Backend Proxies to run client requests. Instances of applications can schedule jobs to the HPC Node instances.

## Interaction View

Below view showcases how a user interacts with NginX for different requests. These include:

* Requesting a scaleup
* Rerouting traffic through LoadBalancer
* Cache for faster delivery of content
* Using the Content Delivery Network to geolocate content and deliver the user low latency experience.

A picture containing text, diagram, parallel, line

Description automatically generated

Figure: Highlights how users interact with components of NginX

# **Problem 6: Architectural Patterns, Principles and Styles of NGINX**

## Patterns, principles, styles, and/or tactics of software architecture used by NGINX

NGINX has a master-slave architecture by supporting event-driven, asynchronous, single-threaded, non-blocking model. [8]

* **Event-Driven**: NGINX employs an event-driven architecture where the main process runs an event loop. This event loop is responsible for handling and dispatching events, such as incoming client connections and I/O operations, in a non-blocking manner. This allows NGINX to process multiple events concurrently, maximizing throughput and responsiveness.
* **Asynchronous**: With asynchronous operations, NGINX can initiate I/O operations, such as reading from or writing to sockets, without waiting for the operation to complete before proceeding. Instead, NGINX continues executing other tasks or handling additional connections while the I/O operation is in progress. Once the operation is completed, NGINX receives a notification or event, which triggers the appropriate callback or continuation for further processing.
* **Non-Blocking**: NGINX utilizes non-blocking I/O operations to handle network communication. By employing non-blocking sockets and I/O multiplexing mechanisms like epoll or kqueue (depending on the underlying operating system), NGINX can efficiently manage multiple connections and perform I/O operations without blocking the execution flow. This enables NGINX to handle numerous client requests concurrently, making it suitable for high-traffic scenarios.

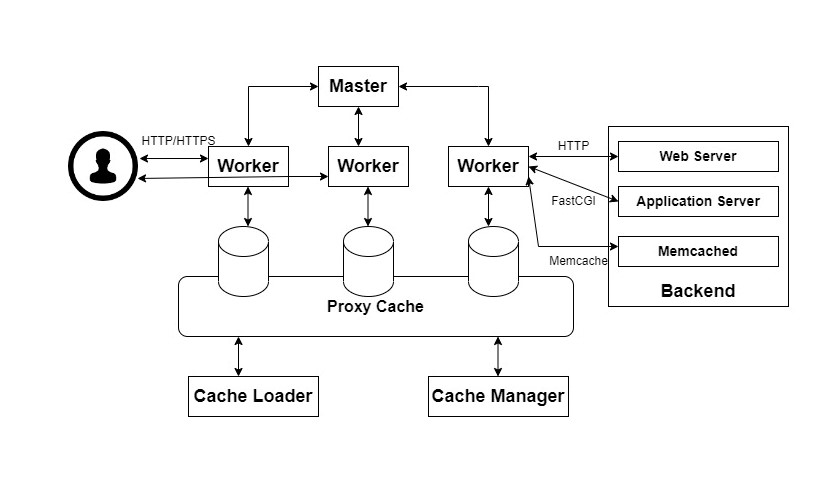
****

Figure: NGINX's architecture.

### Description of NGINX architecture

NGINX was designed with the objective of achieving better performance, resource utilization, and scalability for websites. To accomplish this, NGINX adopted a unique approach influenced by the advancements in event-based mechanisms found in various operating systems. The result was the development of a foundational code that is modular, event-driven, asynchronous, single-threaded, and non-blocking.

NGINX heavily relies on multiplexing and event notifications, and it assigns specific tasks to separate processes. The processing of connections occurs within a highly efficient run-loop in a limited number of single-threaded processes known as workers. Each worker in NGINX has the capability to handle a large number of concurrent connections and process requests at a high rate, allowing for efficient utilization of server resources.

NGINX operates by running multiple processes in memory, including a master process, worker processes, and dedicated processes for cache loading and management. In version 1.x of NGINX, all processes are designed to be single-threaded. To facilitate communication between these processes, shared-memory mechanisms are predominantly utilized.

#### Master process: The master process is executed with root user privileges, while the worker processes, and cache loader run under an unprivileged user.

The master process is mainly responsible for these tasks: [7]

* Create, bind and close sockets.
* Read and validate configuration.
* Initialize, terminate and maintain the configured number of worker processes.

#### Worker Process: NGINX does not create a separate process or thread for each connection. Instead, worker processes handle incoming requests from a shared "listen" socket and utilize an optimized run-loop within each worker to process a large number of connections efficiently. [7] NGINX does not involve any specific distribution of connections among workers as this responsibility is managed by the operating system kernel mechanisms.

During start-up, NGINX establishes an initial set of listening sockets. The worker processes continuously accept new connections, read from and write to these sockets, and simultaneously process HTTP requests and responses. This approach enables NGINX to handle a high volume of connections while efficiently managing resources and maximizing performance.

#### Cache: The cache feature of NGINX allows for rapid page rendering by retrieving content from cache memory instead of fetching it from the server. When a page is requested for the first time, NGINX stores a copy of the page in cache memory. Subsequent requests for the same page can then be served directly from the cache, resulting in faster response times.

#### NGINX Configuration: NGINX configuration simplifies the daily operations and allows for easy expansion of the web server's configuration. The NGINX configuration is stored in plain text files, typically located in "/usr/local/etc/NGINX" or "/etc/NGINX". The primary configuration file is commonly named "nginx.conf". [7]Upon start-up, the master process reads and validates the configuration files. Once the worker processes are created from the master process, they have access to a compiled, read-only version of the NGINX configuration. The configuration structures are automatically shared among the worker processes through standard virtual memory management mechanisms.

## Undesirables of NGINX

**'Undesirables'** in software architecture are code smells and anti-patterns that suggest future design and implementation problems. Code smells are indicators of bad design decisions that could cause issues with performance and maintainability. They serve as cues for programmers to modify or enhance areas of the codebase. Contrarily, anti-patterns appear appealing at first but ultimately have detrimental effects. They are regarded as poor practices and can reduce the software system's quality and scalability. Code smells and anti-patterns both act as markers for spotting and fixing issues with software design [10].

We read through the community-posted tickets on complaints and concerns on the NGINX website to gain an understanding of the unpleasant aspects of the NGINX software architecture. The probable antipattern and code smell in the NGINX source code are discussed below [11].

* Inconsistent cache validation: When a web application or server's cache mechanism fails to consistently update or invalidate cached content as necessary, the term "inconsistent cache validation" is used. Users may receive out-of-date or stale content as a result, raising questions regarding data accuracy and the user experience. Concern occurs when an object is disabled (returns a 404 status) for changes or remakes and then re-enabled in the user-highlighted instance of NGINX caching behaviour. NGINX continues to deliver the cached 404 answer even when the item is now again available, necessitating manual cache clearing to show the updated information. Users may become confused by this inconsistency, experience broken links, and lose faith in the application [12].
* Inconsistent variable configuration: When variables or their configurations are not handled consistently across the codebase, it might result in unexpected behaviour and problems. This is known as an antipattern.The user in this instance is concerned about a variable's handler and value being inconsistent. Using the NGX\_HTTP\_VAR\_CHANGEABLE flag, the variable "sp\_resid" is added and specified in the NGINX module at the preconfiguration step. It is connected to the get\_handler function ngx\_http\_sp\_resid\_variable. The variable is empty when used in a subrequest location block, suggesting that the value is not being set appropriately. The unexpected behaviour is caused by the get\_handler method being overridden, according to further analysis. It is essential to provide uniform variable handling across the whole codebase. Verifying the value and handler of variables, correctly configuring, and updating them, and preventing unintentional overrides or alterations are all part of this [13].
* Subrequest Limitations: When a service or application that uses subrequests encounters restrictions or unexpected behavior because of the absence of support for nested subrequests, this code smell is present. In this instance, the problem occurs while sending a subrequest to a site where proxy\_cache\_background\_update and proxy\_cache\_use\_stale directives have enabled caching. It is assumed that the subrequest will respond right away and, if necessary, serve stale data. The answer is delayed because the subrequest stops each time the cache item needs to be refreshed due to a limitation in nginx's subrequest mechanism. Finding a different strategy that avoids nested subrequests or redesigning the code to handle caching and subrequests in a different way are the suggested solutions [14].
* Insufficient test coverage: The reported problems imply that the current unit tests for nginx configurations with SSL directives and certificate files might be enhanced. The problems include limited testing of SSL directives other than ssl\_certificate, a lack of validation for certificate metadata, and a failure to test alternative algorithms. To fix this issue the user suggested to improve test coverage, validate metadata. This issue is still open for work to be done for the NGINX agent [15].
* Incomplete log Management: This code smell describes a situation where the application or service lacks suitable setup for log rotation, making log management difficult in contexts with long uptimes. In this instance, the Linux packages for nginx-agent do not provide logrotate configuration. As a result, the agent's log file may eventually get very huge, making log management challenging and possibly taking up too much disc space. A logrotate setting for the nginx-agent package is suggested as a fix. The nginx-agent package can improve its log management capabilities, assuring effective disc space utilization and simple log analysis, by fixing this code smell and providing suitable log rotation configuration [16].

# **Problem 7: Suggestions for improving NGINX architecture and architecture description**

## Suggestions for improving Software Architecture Description of NGINX

The evaluation on Nginx Architecture Description is based on “Question Framework for Architectural Description Quality Evaluation”[17]. The evaluation is scored for 10 where suggestion is provided with reasoning where there is a score gap and in case of perfect scores the reason for the score is explained in the Reason/Suggestion section.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Criteria | Questions/ Metrics | Score  (on 10) | Reason/ Suggestion |
| 1 | Stakeholders | Are the stakeholders of the description defined and who are they? | 10 | The stakeholders are represented in the architecture diagram along with a detailed description of their routine tasks [18] |
| 2 | Purpose | Is the purpose of the description defined in relation to the stakeholders? | 10 | The description is provided in accordance with the stakeholders tasks. |
| 3 | Suitability for the stakeholders | Does the description provide the stakeholder with the desired knowledge?  Does the description answer/correspond to the objective of stakeholder?  Does the description relate to problem?  Is a practical reason for the information evident? Is the information presented from the stakeholders’ point of view? | 8 | The description provides knowledge, addresses the objectives and is presented in the point of view of the stakeholder. However, there is a gap in providing performance metrics, benchmarks, or comparison studies that demonstrate Nginx's capabilities in addressing concerns related to scalability, throughput, response times, or resource utilization. This data can help stakeholders assess the suitability of Nginx for their specific use cases. |
| 4 | Usage | Frequency of use: How frequently the description is used or referenced.  Number of users: The approximate number of personnel who will likely want or need to use the description. Variety of users: The variety of different functional areas or skill levels of personnel who will likely use the description. Impact of non-use: The level of adverse impact that is likely to occur if the description is not used properly | 7 | The description provides no clarity in addressing the concerns of the stakeholders, improving the efficiency, reducing the costs, enhancing the security, or simplifying their workflows. |

**Table 1:** Evaluation question framework for the stakeholder and purpose orientation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Criteria | Questions/ Metrics | Score  (on 10) | Reason/ Suggestion |
| 1 | Scope and focus | Scope: Is it defined what part of reality will be described (e.g. only primary processes)?  Aspects: Is it defined what aspects will be described?  The level of detail: Is it defined what level of detail will be described? | 8 | The clarity on scope of the description is vague. For example, the level of detail on the functionalities such as Load balancing, fault tolerance, performance scalability are abstract. |
| 2 | Currency of EA description | 1. Does the information reflect the current enterprise?  2. Were any changes made in the EA after the EA description has been produced?  3. Number and scope of architectural effects having projects carried out after the EA description have been produced 4. Number and scope of architecture changes made after EA description has been produced  5. Degree with which the current version of the description is up to date (Percentage, subjective evaluation)  6. How long is it since the previous updating of the description? | 10 | There are timely updates on documentation, release notes and change logs. The community engagement is active with prioritized considerations of insights and updates from the communities. |
| 3 | Currency of SA description | 1.Does the information reflect the system?  2. Have there been any changes in the system after the architecture description was produced?  3. How long is it from the previous updating? | 9 | The release notes and change logs provide detailed information about the changes made in the system, features added and performance enhancements. Version compatibility is maintained as a cross reference to the architecture description. The frequency of the update is in sync with the development process. However, the details of the performance and scalability testing or comparison with the previous versions after the changes were not presented in the release documents[19]. |
| 4 | Correctness of Information | Verification of information: Is the information included in the description verified?  Are there any incorrect arguments, or in-accurate or untrue reasoning? | 10 | The reasoning provided are cross-referenced with other relevant document sources, feedback of Nginx experts are obtained, as the project progresses, regular reviews and updates on the architecture description to reflect any changes, updates, or lessons learned from implementation and operational experiences are performed. |
| 5 | Correctness of EA | ‘‘Substantive’’ errors/deficiencies after the EA description has been released: Are there ‘‘substantive’’ errors/deficiencies? The number of ‘‘substantive’’ errors/deficiencies found (e.g. the number and type of change request applied to EA principles)? | 10 | The releases indicates substantive errors/deficiencies in the description[18] |
| 6 | Correctness of SA | Correctness for stakeholders: Does the description present correctly the needs and concerns of stakeholders?  Correctness of solution: Does the description define correctly an architecture that will meet stakeholder’s needs? | 10 | The description meets the criteria of correctness of the stakeholders and correctness of the solution[19]. |
| 7 | EA completeness | EA’s coverage of business areas: The degree to which EA description addresses needs of each business area (e.g. subjective evaluation score 1–10) | 9 | The EA description includes understanding the unique requirements, strategic goals and key performance indicators (KPIs). However, clarity on dependencies on technology infrastructure is quite ambiguous[18]. Clarity on this aspect would increase the quality of the description. |
| 8 | Sufficiency/ completeness | Description’s coverage of required viewpoints: The degree to which description addresses each required architectural viewpoint (e.g. subjective evaluation score 1–10). Sufficient amount of information: Is the all required information included in the description? Are all topics relating to stakeholder’s objectives and concerns covered, and only those topics?  Is information repeated only when needed?  Does the description contain irrelevant or superfluous elements?  Sufficient level of detail: Has each topic has just the detail that stakeholder needs? | 10 | The description addresses identifying the view points, mapping the architecture to the viewpoint, identifying the gaps, expanding the coverage, collaboration with stakeholders with iterative review and improvements. |
| 9 | Consistency | Are views presenting different viewpoints in the description consistent with each other? | 10 | The viewpoints, dependencies and information presented in the document are consistent with each other. |

**Table 2:** Evaluation question framework for the content

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Criteria | Questions/ Metrics | Score  (on 10) | Reason/ Suggestion |
| 1 | Conformance to corporate standards | Does the presentation of the description conform to the corporate standards (if any) for such documents? | 10 | The description meets the criteria of general guidelines that are often considered when presenting architecture descriptions. |
| 2 | Intuitiveness of the presentation | Does the description have an intuitive structure for the stakeholder?  What is this intuitive structure? Does the description correspond to it? Is the recipient familiar with the structures used? | 9 | The description meets the criteria of Intuitiveness of the presentation except for the future roadmap which outlines the upcoming upgrades for the architecture. |
| 3 | Definition of the notation and structures | Does the description use a defined notation?  Is the notation/structure of the description explained?  Is stakeholder familiar with notation? | 8 | The Nginx architecture documentation typically uses a combination of text, diagrams, and sometimes code snippets to describe the architecture and its components.The familiarity of stakeholders with the notation used will vary depending on their background, experience, and level of familiarity with architectural documentation. Technical stakeholders, such as developers or system administrators, may have a higher likelihood of being familiar with the notations commonly used in architectural descriptions. However, it's important to consider the audience and ensure that the notation and structure of the description are explained clearly and comprehensively to accommodate stakeholders who may be less familiar with technical notations. |
| 4 | Clarity of the vocabulary and concepts | Are the terms and concepts used known by the stakeholder?  Are the terms used defined? Are the (new) concepts defined and explained?  Are the names of elements descriptive?  Are all of the description’s elements defined so that their meanings, roles, and mapping to the real world are all clear and not open to different interpretations? | 10 | The description meets the criteria of Clarity of the vocabulary and concepts. |
| 5 | Information complexity | Is there too much information included in the model?  The number of elements in the model. (Humans are only good at working with models that do not include more than 30 elements.)  The number of types of elements in the model.  The number of relations depicted in the model.  The number and types of concepts.  The number of architectural viewpoints. (Viewpoints reduce complexity). | 10 | There is very elaborate and necessary information with less than 30 elements included for this model. The types of elements and the relations depicted are less and have minimal viewpoints. |
| 6 | Visual complexity | Proximity: Are the related objects placed near to each other in a model?  Continuity: Are there any right angles positioned next to each other? (Right angles should not be positioned next to each other in a model.)  Closure: Are objects symmetric and regular? (This increases readability of models and reduces the perceived complexity.)  Similarity: Are similar objects presented in the similar way? Common fate: Are similar object presented to move or function a similar manner? (People have a tendency to perceive different objects that move or function in a similar manner as a unit.) | 10 | Objects placed near as needed in this model are clear enough to give a proper continuity throughout the flow[18]. |

**Table 3** Evaluation question framework for the presentation and visualization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Criteria | Questions/ Metrics | Score  (on 10) | Reason/ Suggestion |
| 1 | Maintenance of documentation | Ownership:  Are the staffs responsible for the documentation clearly identified and supported?  Maintenance practice:  Is it known how the documentation will be maintained once it has been accepted?  Is the frequency of updating known?  Frequency of updates (number of updates/year or project). Needs for updates (number of architecture changes made in a year, in projects that require documentation update). Maintainability of documentation:  The relative ease or difficulty with which the documentation can be updated, including revision dates and distribution of new versions and the relative ease or difficulty with which the consistency between descriptions can be checked | 10 | The release notes and change logs have clear information about the ownership of the document. Frequency of update is maintained with versioning. The maintainability standard is high with respect to the documentation[19]. |
| 2 | Cost effectiveness | Costs: Time and resources needed to produce or update architecture documentation (required man-days).  Amount of documentation: Number of documents/models. Frequency of documentation updates: Updates/project or updates/year. Needs for updates (number of architecture changes made in (a year, in projects) that require documentation update | 10 | Very frequent releases in order to resolve issues from previous releases, bug fixes, and security updates. This also increases the frequency of documentation by the same amount. |
| 3 | Architectural framework and views | Architecture framework (for EA and for SA):  Is there existing architectural framework?  Is the framework accepted in organisation?  Is the framework used in the EA documentation work? Architectural views: Are the suitable architectural views chosen for the company or for the project?  Are the viewpoints well defined?  A Viewpoint name?  The stakeholders the viewpoint is aimed at?  The concerns the viewpoint addresses?  The language, modelling techniques, or analytical methods to be used in constructing a view based upon the viewpoint? | 10 | The viewpoints are clearly defined with appropriate names in a way stakeholder’s would understand and relate to the use case[19]. |
| 4 | Tools support | Support for organisation’s framework and viewpoints:  Do design tools support the framework and viewpoints that organisation has chosen to use? Do design tools support production of the deliverables required?  Suitability for Stakeholders: Is there ability to represent architecture descriptions (e.g. models and views) in a way meaningful to stakeholders (e.g. to non-technical stakeholders)? Repository for architecture documentation: Is there a repository for storage and dissemination of the captured information? | 10 | Nginx supports the organization's framework, viewpoints and organization's specific requirements and standards.  The centralized documentation repository of Nginx provides a single source of truth, version control, and access control mechanisms. It allows stakeholders to access and review the architecture documentation when needed and ensures the availability of up-to-date information. |

**Table 4:** Evaluation question framework for the architecture documentation management

## Suggestions for Improving Software Architecture of NGINX

### Performance Optimization

1. Architecture can be modified to utilize hardware resources efficiently, such as CPU and memory, by adjusting worker processes, worker connections, and other related settings.
2. Implement caching mechanisms to reduce the load on backend servers and improve response times for static content.
3. Fine-tune load balancing configurations to evenly distribute traffic and optimize resource utilization.
4. Utilize connection pooling to reuse connections and minimize the overhead of establishing new connections.

### Security Enhancements

1. A WAF module or a dedicated WAF solution can be integrated to protect against common web application vulnerabilities and attacks.
2. DDoS protection mechanisms, such as rate limiting, request filtering, and IP whitelisting/blacklisting can be implemented to mitigate the impact of distributed denial-of-service attacks.

### Monitoring and Logging

1. Monitoring tools can be integrated to track performance metrics, component’s health, and resource utilization in real-time. This enables proactive identification of potential issues and facilitates capacity planning.
2. Detailed logs can be generated to include access logs and error logs, for troubleshooting, security analysis, and auditing purposes. Centralizing and analyzing the logs using log management tools to gain insights into the system's behavior.

# Team Contributions

|  |  |
| --- | --- |
| **Group/Team/Unit Member** | **Contributions** |
| Abhimanyu Sharma | * Identified and analyzed the Quality Attributes that NGINX aims to satisfy (Problem 4). * Identified and analyzed the Quality attributes that NGINX should satisfy but does not (Problem 4). * Formatted the document |
| Kirthana Senguttuvan |  |
| Saurabh Sharma |  |
| Urvish Rupeshbhai Tanti | * Researched and analyzed community posted tickets on the NGINX website and github (Problem 6b) * Identify code smells and anti-patterns in the NGINX software architecture (Problem 6b) * Helped formatting the document |
| Vaibhav Sharma |  |

# References

[1] INTRODUCTION TO SOFTWARE PRODUCT QUALITY BY PANKAJ KAMTHAN <http://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_product_quality_introduction.pdf>

[2] NGINX Features <https://nginx.org/en/>

[3] NGINX- Frequently Asked Questions <https://www.nginx.com/resources/faq/?_ga=2.223001819.1799938747.1687105030-473675994.1687105030>

[4] NGINX documentation <https://nginx.org/en/docs/>

[5] The Architecture of Open Source Applications (Volume 2) nginx-Andrew Alexeev <https://aosabook.org/en/v2/nginx.html>

[6] <https://docs.nginx.com/nginx-service-mesh/about/architecture/>

[7] <https://aosabook.org/en/v2/nginx.html>

[8] <https://medium.com/@premsuryamj/nginx-architecture-9f97cf7887e2>

[9] <https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/>

[10] <https://users.encs.concordia.ca/~kamthan/courses/soen-6471/software_architecture_undesirables.pdf>

[11] <https://trac.nginx.org/nginx/report>

[12] Proxy cache 404 STALE forever <https://trac.nginx.org/nginx/ticket/1614>

[13] The get\_handler of ngx\_http\_variable\_t is overwritten by ngx\_http\_regex\_compile if existing <https://trac.nginx.org/nginx/ticket/2213>

[14] "proxy\_cache\_background\_update on" ignored using subrequest (more exactly: nested subrequest) <https://trac.nginx.org/nginx/ticket/2389>

[15] Increase coverage in NGINX config unit tests <https://github.com/nginx/agent/issues/276>

[16] nginx-agent packages do not have logrotate configuration <https://github.com/nginx/agent/issues/310>

[17] N. Hämäläinen and J. Markkula, “Question framework for architectural description quality evaluation,” *Software Quality Journal*, vol. 17, no. 2, pp. 215–228, Jan. 2009, doi: <https://doi.org/10.1007/s11219-008-9068-1>.

[18] “Architecture Overview | NGINX Documentation,” *docs.nginx.com*. <https://docs.nginx.com/nginx-management-suite/acm/about/architecture/>

[19] “NGINX Management Suite | NGINX Documentation,” *docs.nginx.com*. <https://docs.nginx.com/nginx-management-suite/>.