Cloud Computing: Software Server Architecture for Distributed System

Binoy Kurikaparambil Revi

Independent Researcher, Tennessee, USA
Email: *binoyrevi[at]live.com*

Abstract: ***Software development of on - prem or cloud applications often encounter requirements where the application modules need to be deployed on a distributed system or run as separate processes in a single server. This is done mainly to leverage the hardware performances, keeping the software modules highly encapsulated and making the software highly scalable and configurable. Software Server Architecture for Distributed Systems provides an excellent framework to develop software solutions for distributed systems or a multi - process application on a single system.***

Keywords: Distributed System, Kubernates, Cloud Computing, Software Servers

1. Introduction

Software Server Architecture for distributed systems uses the processes running across multiple systems to communicate with each other using secure communication services like Restful Services. This means all the configurations, data, messages, status etc. shall be exchanged between the processes using Rest APIs. If the processes are running on a single system, socket communication is an option, however Restful Services are preferred to easily support the high - level programming like JavaScript and can be enhanced to be more secure. Restful services enable the application processes to communicate with each other efficiently with a well - defined protocol and JSON data model. The implementation of the application doesn't necessarily depend on where the application is expected to run. It can run on a single system, or it can run on a distributed system depending on the complexity and design of the application. This architecture basically brings the idea “Write Once, Run Anywhere”.

2. High Level Architecture

Software Server Architecture can cater to single server design or distributed system design. In both cases the system can use on - prem servers or cloud servers depending on the system requirements. The application is run using the software processes running on these servers which are called as Software Servers. [2] Name contains the term server as it can accept HTTP requests and respond to these requests as part of Restful services. The Software Servers also send the request to other Software Servers that are an integral part of the application. Data Model designed to exchange data between the servers and manage internal data plays a critical role in this architecture. A typical example of the Software Server Architecture on a single server is given in figure 1.

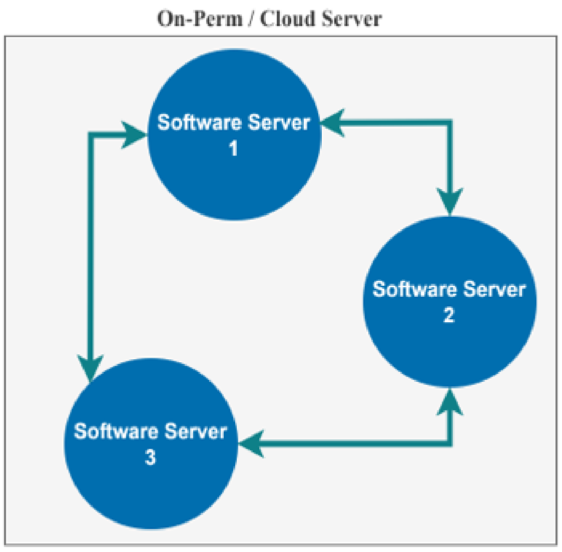

Figure 1: Software Servers running on a single server

On a multi - server or distributed architecture, the software servers run on different servers [2] and communicate with each other using Restful Services. Figure 2 describes a high - level architecture of software servers on distributed systems.

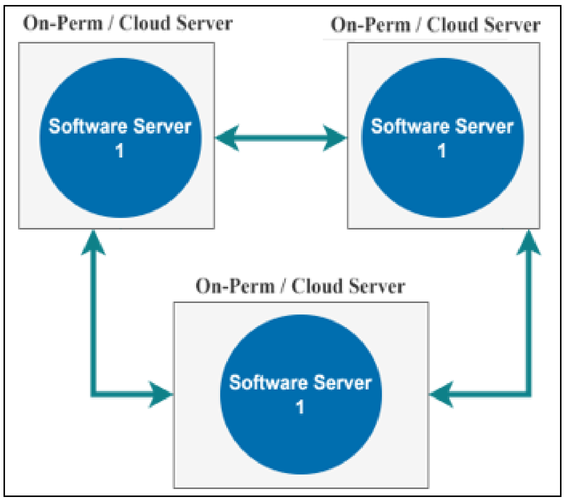

Figure 2: Software Servers running on distributed system

3. Cloud Application Design using Software Servers

In the modern - day application deployments, dockerization or containerization is considered as a preferred strategy as there are less chances of surprises that come out during deployment and execution of the application in production. This is because the containerization provides the exact well - defined environment in the production as in the development for the application to run. The container virtually acts as the complete system on which the application can run as it is running on an expected hardware with an operating system. [4] Docker has become the de facto standard for containerization and Kubernetes which is an open - source system that automates the management, scaling, and deployment of containerized applications has become the preferred model to manage the containers.

Software Servers Architecture can be implemented by containerizing the software servers in multiple containers and managing them using the Kubernetes services. Most of the major cloud providers provide a wide range of tools and services to deploy and manage the containers in their Kubernetes service. [4] It is simple to understand how the Software Server Architecture works in the container world. Each application module runs on one container and uses its IP address and specific port number to communicate with other containers using restful services. However, it is also possible to run more than one Software Servers in one server and remaining Software Servers on other servers. Figure 3 provides a typical example of Software Servers Architecture on cloud.

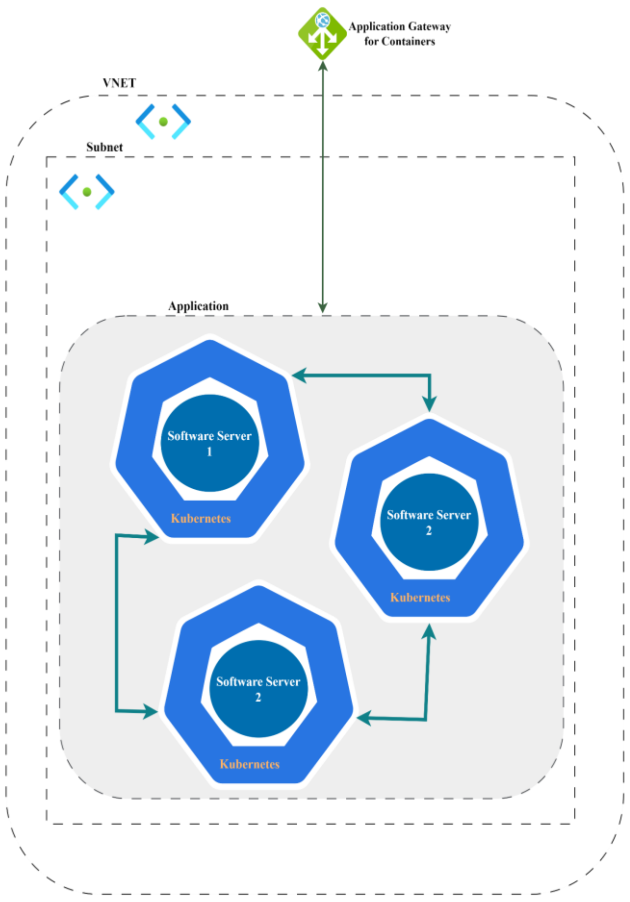

Figure 3: Cloud Software Server Architecture

4. Implementation of Software Servers Using QT Framework

Figure 4 describes the implementation of Software Server Architecture using the QT framework. The intention of picking the QT framework to build the software servers using C++ is because in most real - world applications the backend core application that serves as the brain of the application is mostly written in C++ or something similar. QT framework provides QT libraries that can be used to add Restful services to the C++ applications. This can be run as a software thread in the C++ application and has proven very effective and efficient.

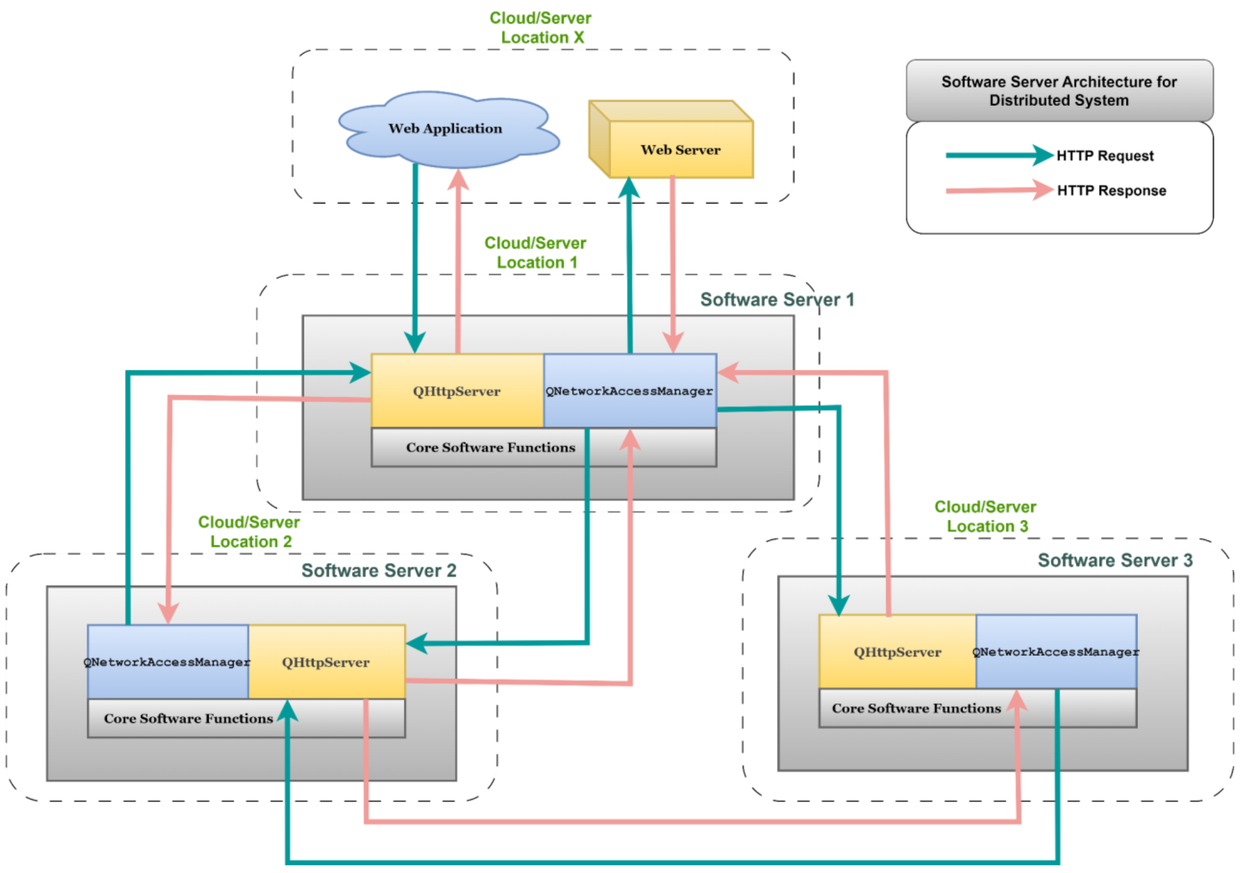

Figure 4: Implementation of Software Servers

Using the QT framework, the Software Server is built by attaching each application module with QNetworkAccessManger and QHttpServer classes. QHttpServer helps build HTTP server functionality into an application. This allows the Software Server to receive Restful requests over HTTP and respond to the request. The QNetworkAccessManager class allows the application to send network requests and receive replies allowing Software Servers to send Restful requests to other Software Servers and receive responses. All the asynchronous functionalities provided by these classes are managed by QT Signal Slot mechanism that enables non - blocking execution of the application modules. Each Software Server can run as a separate process in the machine or in the container environment and communicate with other Software Servers using the Restful Service using the QNetworkAccessManger and QHttpServer classes. QJsonObject is another class that comes in handy to manage and process the JSON data. JSON Data model is the preferred data model for the Software Server Architecture as it is more easy to use with built in libraries from QT, serves as a standard data model for latest JavaScript frameworks like React and Vue Js and it is more secure than traditional XML.

5. Overhead of Software Server Architecture

The Sections above talk about the idea, design and implementation of the Software Server Architecture, however it is also important to analyse the implementation overhead that comes with this architecture. This architecture is best suited to relatively highly complex and large software solutions where the solution requires multiple software modules to solve a complex problem. Architecture may not be the best choice for a single major software module application. Other overheads that come during the implementation are described below:

1) QNetworkAccessManger and QHttpServer classes - These classes are added to every Software Servers to enable communication.
2) Application Health Management Server [1] - This is one of the main and common use cases when designing the application using Software Server Architecture. The Health Management Server runs as a Software Server itself, is responsible for the initialization of other software servers and responsible for checking the health and status of all Software Servers.

6. Conclusion

Software Server Architecture provides a framework to design big and complex software solutions as independent modular solutions which can be efficiently built, maintained and scaled irrespective of the technology or programming language that is used to build each module. This framework enables different teams to work on their area of expertise to build the software servers that can serve a part of the complete solution without knowing anything inside other modules. Only dependency is the data model that is agreed for the communication between Software Servers.

References

[1] M. Sloman, J. Magee, K. Twidle and J. Kramer, "An architecture for managing distributed systems, " 1993

4th Workshop on Future Trends of Distributed Computing Systems, Lisbon, Portugal, 1993, pp.40 - 46, doi: 10.1109/FTDCS.1993.344178.

[2] H. Gomaa, "Advances in Software Design Methods for Concurrent, Real - Time and Distributed Applications, " 2008 The Third International Conference on Software Engineering Advances, Sliema, Malta, 2008, pp.451 - 456, doi: 10.1109/ICSEA.2008.78.

[3] M. S. A. Muthanna and A. Tselykh, "Development of Docker and Kubernetes Orchestration Platforms for Industrial Internet of Things Service Migration, " 2022 International Conference on Modern Network Technologies (MoNeTec), Moscow, Russian Federation, 2022, pp.1 - 6, doi: 10.1109/MoNeTec55448.2022.9960769.

Author Profile

Binoy Kurikaparambil Revi received a bachelor’s degree (BTech) in Electronics and Communication Engineering from Cochin University of Science and Technology in 2004. He received a Master of Science (MS) in Computer Science with concentration in Cybersecurity from University of Tennessee, Knoxville in 2024. He has 19 years of experience in software development in various domains that include Aerospace, Medical Devices, Industrial Electronics and Energy Solution. Currently he works as a Senior Software Engineer at PXiSE Energy Solution, San Diego, California.