2014

A Portfolio View of a Microsoft Enterprise Architecture

Jerod Gawne
Grand Valley State University

A Portfolio View of a Microsoft Enterprise Architecture

By
Jerod Gawne
December, 2014

A Portfolio View of a Microsoft Enterprise Architecture

By
Jerod Gawne

A project submitted in partial fulfillment of the requirements for the degree:

Master of Science in
Computer Information Systems

At
Grand Valley State University
December, 2014

Gregory B. Schymik, Ph.D.

Contents

Abstract

An enterprise architecture (EA) establishes the organization-wide roadmap to achieve an organization's mission through optimal performance of its core business processes within an efficient information technology (IT) environment. Simply stated, enterprise architectures are "blueprints" for systematically and completely defining an organization's current (baseline) or desired (target) environment (Schekkerman, 2011).

If defined, maintained, and implemented effectively, these blueprints assist in optimizing the interdependencies and interrelationships among the business operations of the enterprise and the underlying IT that support these operations. It has shown that without a complete and enforced EA (Strategic) Business Units, the enterprise run the risk of buying and building systems that are duplicative, incompatible, and unnecessarily costly to maintain and interface.

While all the perspectives are key elements of the enterprise architecture, the focus of this project is scoped to three EA perspectives. The first two perspectives are EA's application and technology architectures, their concepts and key patterns for construction of service oriented and message-based applications that exploit the emerging technology of asynchronous communication services. The third perspective covered is EA's implementation perspective, which includes the design, development, setup, deployment, and administration of enterprise systems that enable enterprise architecture and modern agile software development and project management.

Keywords: enterprise architecture, software as a service, asynchronous, networking protocols

1. INTRODUCTION

Enterprise Architecture

As defined by ANSI/IEEE Std. 1471-2000 (IEEE 1471) Recommended Practice for Architecture Description of Software-Intensive Systems, enterprise architecture (EA) is "the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution." (ANSI/IEEE, 2001) The primary goal of an EA is to "translate business vision and strategy into effective enterprise," (Gartner, 2013).

Although terminology differs from framework to framework, many include at least a distinction between a business layer, an application layer, information layer, and a technology layer. Enterprise architecture addresses among others the alignment between these layers, usually in a top-down approach.

EA is a conceptual tool that assists organizations with the understanding of their own structure and the way it works. It provides a map of the enterprise and is a route planner for business and technology change. Typically an EA takes the form of a comprehensive set of cohesive models that describe the structure and the functions of an enterprise. Important uses of it are in systematic IT planning and architecting, and in enhanced decision making.

The individual models in an EA are arranged in a logical manner, and this provides an ever-increasing level of detail about the enterprise, including:

- Its objectives and goals
- Its processes and organization
- Its systems and data
- The technology used

Architectural Perspective

The information in the enterprise architecture can be viewed from many perspectives and it can satisfy many needs. Architectural users include business managers and analysts, system architects and designers, workflow and procedures analysts, logistics specialists, organizational analysts, and so on. These people require high-level summary information, detailed data, and all levels in between. These demands are met through the creation of conceptual views, logical analyses, and physical implementations (Platt, 2002).

Research shows that four general perspectives are important and are commonly used. These are the business, application, information, and technology perspectives.

The Business Perspective

The business perspective describes how a business works. It includes broad business strategies along with plans for moving the organization from its current state to an envisaged future state. It will typically include the following:

- The enterprise's high-level objectives and goals.
- The business processes carried out by the entire enterprise, or a significant portion of the enterprise.
- The business functions performed.
- Major organizational structures.
- The relationships between these elements.

The Application Perspective

The application perspective defines the enterprise's application portfolio and is application-centric. This view will typically include:

- Descriptions of automated services that support the business processes.
- Descriptions of the interaction and interdependencies (interfaces) of the organization's application systems.
- Plans for developing new applications and revising old applications based on the enterprises objectives, goals, and evolving technology platforms.
- The application perspective may represent cross-organization services, information, and functionality, linking users of different skills and job functions in order to achieve common business objectives.

The Information Perspective

The information perspective describes what the organization needs to know to run its business processes and operations. It includes:

- Standard data models.
- Data management policies.
- Descriptions of the patterns of information production and consumption in the organization.

The information perspective also describes how data is bound into the work flow, including structured data stores such as databases, and unstructured data stores such as documents, spreadsheets, and presentations that exist throughout the organization.

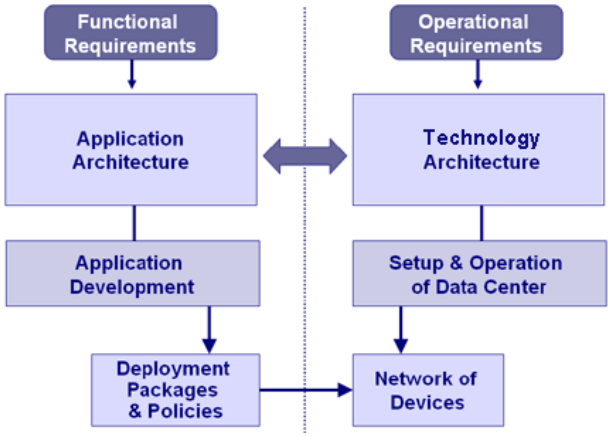

Figure 1 Relationships between

The Technology Perspective

The technology perspective lays out the hardware and software supporting the organization. It includes, but is not limited to:

- Desktop and server hardware.
- Operating systems.
- Network connectivity components.
- Printers.
- Modems.

The technology perspective provides a logical, vendor-independent description of infrastructure and system components that are necessary to support the application and information perspectives. It defines the set of technology standards and services needed to execute the business mission.

Application and Technology Architecture

The functional requirements of a software system describe the business value that the software delivers. For a communication service, a functional requirement might be stated as "given well-formed message 'A' as input, the service will return message 'B' correct for the intended recipients as represented in message A."

An application architecture is the architecture of any automated services that support and implement such functional requirements, including the interfaces to the business and other applications. It describes the structure of

an application and how that structure implements the functional requirements of the organization. Whilst there should ideally be one application architecture in an organization, in practice there are typically many different application architectures.

The operational requirements of a software system define the reliability, manageability, performance, security, and interoperability requirements of the software (to list just a few). Common examples might be that the service is only available to authorized subscribers, and that the service be functioning properly 99.999 percent of the time.

In order to arrive at the operational rules of a software system requirements gathering must take place. However, requirements gathering is only half of the work involved in requirements specification. Translating those ideas into features, back-log items, and tasks in enough detail to provide the developer with the requisite instructions and without spending a lot of time can be difficult. When I began this project I started in a very agile fashion by loosely defining product features and given that I didn't know C# set forth to start implementing. About half way through the project I began to understand what it takes to define software requirements. Writing out what I thought the app should do and then actually coding it I learned how to better write user-stories and acceptance criteria.

A technology architecture is the architecture of the hardware and software infrastructure that supports the organization and implements the operational (or non-functional) requirements, particularly the application and information architectures of the organization. It describes the structure and inter-relationships of the technologies used, and how those technologies support the operational requirements of the organization.

A good technology architecture can provide security, availability, and reliability, and can support a variety of other operational requirements, but if the application is not designed to take advantage of the characteristics of the technology architecture, it can still perform poorly or be difficult to deploy and operate. Similarly, a well-designed application structure that matches business process requirements precisely—and has been constructed from reusable software components using the latest technology—may map poorly to an actual technology configuration, with servers inappropriately configured to support the application components and network hardware settings unable to support information flow. This demonstrates the relationship between the application architecture and the technology architecture.

In the past, applications have been built by integrating local system services such as file systems and device drivers. This model was very flexible in providing access to a rich set of development resources and precise control over how the application behaved; however, this was very error-prone, costly, and time-consuming.

Today, complex distributed applications are being constructed that integrate existing applications and services from all over their networks and then add unique value on top by using elements such as business entities, data entities, and façades. This enables developers to focus on delivering unique business value. The result is reduced time-to-market, higher developer productivity, and ultimately, higher-quality software. This has been a powerful architectural model for a number of years; however, it creates "application stovepipes" or "islands of information" that cause significant problems in architectural reuse (Platt, 2002).

Each of the elements in the application model requires mapping to elements of real technologies. In this way application models are realized as implementation models. Part of this task is undertaken during conventional development when programmers write detailed business logic as code, but much of the implementation activities are properly classed as framework completion—a technique for development where much of the infrastructure of distributed applications and data management is handled by sophisticated frameworks that are extended by custom application logic and declarative control structures. Framework completion shields developers from the intricacies of, for instance, asynchronous message handling, and allows developers with modest skills to make effective contributions to the project.

Architecting and building these models for an organization at each of the different levels is clearly a considerable amount of work and effort. Additionally, the correct definition of these models is critical for an organization. An incorrect architectural model almost always results in serious design or operational issues such as scalability or reliability problems or, in the worst cases, project non-completion and business impact. Architects are looking for frameworks and roadmaps to assist them in creation and implementation of these models and to minimize the risks associated with the use of incorrect models.

There are two main types of architectural guidance and assistance that can be provided to architects to speed up model generation and minimize risk.

The first of these is a set of architectural concepts that provide:

- A common understanding and communication.
- Guidance as to how and when specific concepts should be used, and information about their attributes.
- An indication as to when these concepts will be realized and available, either in terms of guidance or real technology.

The second, which the development portion of my project implements, is a set of patterns, based on real-world experience harvested from a large number of successful distributed applications, which are composed from these fundamental concepts. These patterns encapsulate important best practices for distributed application design and minimize the risk of project failure by providing known good, tested architectural models.

Application Patterns

A pattern is a solution to a problem in a context. A pattern codifies specific knowledge collected from experience in a domain. An application pattern is an architectural-level design that defines best practices in architectural design for a specific application environment.

A key requirement for organizations is the integration of these disparate technology architectures into one all-encompassing architecture to allow the reuse of present-day applications and the rationalization of these technology architectures to a minimum set. The provision of this single, common architecture is vital to the creation of efficient, effective, and flexible organizations.

The technology conceptual architecture is the technology basis for achieving enterprise-scale services in an organization. The high-level diagram of the technology conceptual architecture shows a set of generic levels that provide enterprise-based services for service generation. These levels contain the common elements that are required by any service application or system. At the bottom of the diagram is the Service Platform, providing operating system, hardware, storage, networking, and the trust and management services for the whole system.

The Service Framework hosts the process, logic, functions, and state management required by a Web service–based application and is the full Enterprise Application Server with specific support for Web services.

Figure 5. Conceptual view of technology architecture. (Sourced MSDN)

Service Delivery contains the portal and client services that focus on presentation issues and technologies, including support for all types of devices.

Service Integration provides integration and interoperation between services and present-day operational systems: legacy applications, commercial applications, databases, and other Web services. This is commonly called Enterprise Application Integration (EAI).

Finally, Service Creation provides the tools, process, methodologies, and patterns required to design, develop, assemble, manage, deploy, and test Web services.

Logical view
The logical view of the technology architecture is where the major functional elements that provide support for enterprise-scale operational requirements and their interrelationships are provided. Enterprise technology elements such as databases, mail systems, transaction support, and reliable messaging are provided in the logical view. The technologies that are provided at this level are normally packaged together as servers by enterprise software vendors.

Technology Patterns
A technology pattern is an architectural-level pattern that defines the best practice architectural design for a specific technology environment. Enterprise architects should refer to best practice for their organizations in the following critical areas:

- Security, identity, and trust.
- Integration and interoperation, both with future systems and present day operational (legacy) systems.
- Deployment, administration, and management.
- Distributed technology architectures for scalability and performance.
- Technology architecture for reliability and availability.

2. BACKGROUND AND RELATED WORK

The value of an enterprise architecture (EA) is not in any one individual perspective, but in the relationships, interactions, and dependencies among perspectives (Platt, 2002). For EAs to be useful and provide business value, their development, maintenance, and implementation should be managed effectively and supported by tools (Schekkerman, 2011).

There are many different product suites, platforms, and topologies out in the market to choose from. Many people are trying to get away from Microsoft products and are going to what they think is 'Open-Source'. Platforms based around Linux such as RedHat, Suse, or Oracle are among the top 'Open-Source' enterprise operating platforms. Linux carries with it a license that prevents anyone who uses the source from charging money for any products derived from that code. Which sounds great, and many people wishing to reduce infrastructure costs are quick to jump off the Microsoft ship thinking that they can easily switch to a 'free' open-source platform. This way of thinking is, however, entirely incorrect.

Among the top operating platforms, only one is actually free. Oracle's implementation uses RedHat's source as a base and primarily maintains it so that they can guarantee compatibility with the host of server applications they sell. RedHat and Suse on the other hand have an interesting scheme in place to charge for essentially free software. In order to charge for their software they offer to give you the software for free along with a costly support contract. While some may think this is a 'this or that' situation and that they are happy to have support, this is only the first cost of open-source platforms.

So great, you've got an open-source operating platform and even if you went with a completely free package what are you going to do with it? The platforms usually come with an ecosystem of free server applications that you can use to setup basic infrastructures, however, the technologies used require quite a bit of know-how and often frustrating configurations in order to get them to work with all the devices on the network. This also requires people that have expert knowledge of Linux platforms.

After finally getting a basic infrastructure setup you now have to implement the major server applications such as an SQL server. Sure there are free SQL servers that will probably work quite well for smaller enterprises, however the major SQL server implementations (that work on Linux) IBM's DB2 and Oracle's 11g or 12c cost a significant amount of money which is added to the amount paid for Microsoft SQL Server if using that previously. Delving into the enterprise application eco-system is beyond the scope of this project, however, of note is that the server applications available on Linux platforms are also available for Windows platforms while the opposite doesn't stand.

3. PROGRAM REQUIREMENTS

The primary reason for choosing the Microsoft platform is the richness, and guaranteed interoperability of the Microsoft server ecosystem. Microsoft has operating platforms, server applications, and user applications all designed in concert that facilitate relationships, interactions, and dependencies among enterprise architecture perspectives that provide business value.

The second reason for choosing the Microsoft platform is its alignment with current job-market requirements. Among the lessons learned from my job-search I discovered the most prolific set of skills sought after by employers were connected to Microsoft platforms. According to (CyberCoders Team, 2013) an I.T. Staffing firm the top 10 most in-demand jobs for 2014 are:

1) Java / JavaScript
2) C# / ASP.NET
3) C++
4) Python
5) PHP
6) SQL / MySQL
7) HTML5 / CSS3
8) Ruby on Rails
9) Hadoop
10) iOS / Android

The final and most practical reason is because it was available to me with no direct cost. Having access to DreamSpark as well as an MSDN subscription through my recently obtained job I have access to more Microsoft products than I can find a use for.

Project Goals

- To acquire a deeper knowledge and experience pool containing more relevant and modern technologies, platforms, and methodologies utilized by enterprises today.
- To enhance theoretical knowledge gained from formal education via translation to associated skills and experiences.
- To create an enterprise class development environment including hardware infrastructure, servers, server software, project management, team collaboration and source control, and lastly software development.
- To create artifacts as proof of experience to be used in a portfolio demonstrating a broad range of capabilities.

Project Objectives

- Learn Hardware[1] Server and Networking Topology in support of n-tier application architecture (physical and virtual as Windows Server and Cloud as Microsoft Azure).
- Learn Windows Server 2012 r2 deployment, configuration, and administration of server roles required to enable server software (Active Directory, DNS, IIS, App Fabric, PowerShell).
- Learn Microsoft SQL Server 2014, SharePoint 2013, and Team Foundation Server 2013 installation, configuration, and administration.
- Learn how to integrate Microsoft Azure instances with on-site physical and virtual infrastructures.
- Learn source controlled, collaborative, and agile (SCRUM) team development using Microsoft Visual Studio 2013.
- Develop an application development project utilizing Team Foundation with the SCRUM methodology.
- Learn business analyst, quality assurance, and development roles while developing the project and creating an application written in C#.

4. IMPLEMENTATION

Although there can be many perspectives, there is only one enterprise architecture that the perspectives view. The value of the EA is not in any one individual perspective, but in the relationships, interactions, and dependencies among perspectives. While all the perspectives are key elements of the enterprise architecture, this paper will focus on the application and technology perspectives. In order to fill an experience gap and open up a broader array of jobs I ventured an ambitious project in order to facilitate a more specific understanding of the connected parts of an enterprise.

Premise

In today's I.T./I.S. landscape there are a few prominent employee types. Specifically,

[1] Physical and/or Virtualized.

concerning how specialized or generalized their education, skillset, and experience is relative to their job title, role, or function. This relationship is directly influenced by the complexity of I.T. and business operations in today's global economy.

There are too many relationships to explore in the time allotted and any concrete research is beyond the scope of this paper; I will instead focus on two easily identifiable archetypes.

1.1.1.1 Highly Specialized

- These employees are becoming increasingly difficult to manage as their work can be too complex for managers to understand, requires years of experience and or education to fully understand, or in combination with the previous two has intrinsic recency requisites to maintain relevancy.

1.1.1.2 Highly Generalized

- The scope of this area is the primary management difficulty. Depending on the size of the enterprise the specific area of difficulty can take varying forms, however, the underlying issue of scope still persists. In small enterprises there might only be one or a few managers to handle many differing roles that are filled with employees that have similar experience in enough quantity to be seen as trainable. Finding a manager that knows enough about several different roles to be able provide guidance to employees as well as translate issues or progress to upper management could be a daunting task. Large Enterprises might have teams dedicated to each role, with one or more managers each. This can add to the complexity of communication, disconnecting the business roles from the implementers of the project.

Given the complexity of today's I.T. and business landscape I conclude that it takes a broad educational and experiential background combined with a functional knowledge base and understanding of how the parts interact with each other in order to successfully manage I.T..

Motivation

In the past I.T. firms would hire computer science and similar degree college graduates straight out of college and with little or no experience and then teach them the skills they needed to fulfill the role they were hired into. Being a candidate for my third degree in four years (two Computer Information Systems, one Business Administration, along with a minor in Communication, A+ certification, and continuing education) I thought myself a desirable candidate for entry level I.T. positions. Quite wrong I was as business has, out of necessity, learned to adapt I.T. for alignment with today's business practices, specifically agile/lean/on-demand processes.

The job requirement I found to be most prolific was for jobs labeled as 'entry level' or had a 'jr.' prefix stated as a mandatory requisite as having one to three years of experience. Which is of course interesting given the 'entry-level' notation as a reasonable person could conclude that to mean zero to three years of experience, especially for college graduates.

A year worth of internships could work but still seems a bit steep of a requisite for someone with plenty of work experience (just not in relevant areas) and can be hard to get. Many internships require application a year in advance and are limited to specific class-standing (sophomore/junior) and some go as far as to specify undergraduate only. This is especially difficult when you earn 180 semester credits in four years; I hit junior standing (in undergrad) after 38 weeks.

I did find, however, that a few employers were willing to entertain portfolios of projects that demonstrated ability, experience, or knowledge. I had those items available, however, when I started school Java and PHP were the prolific technologies, by the time I found myself looking for employment all of the jobs (at least the ones I came across) were for .Net technologies (C#, VB, ASP) and a few for mobile but those were experienced to expert level.

Application & Technology Implementation

To summarize my project; I planned to implement an n-tier server architecture that supported an agile methodology (SCRUM) and team-based project management approach to

application development. The primary focus was around designing and developing the enterprise architecture implementation. Secondary was to gain experience using team collaborative, project management, and versioning systems based around the .Net framework, specifically c#. Tertiary was to design and develop a networking library framework using C# and the windows forms approach to not only learn C# but provide proof that the EA architecture was working as intended.

I've had experience installing software and working with computers and systems for many years, consequently, I inferred from the beginning that this would involve a fair amount of learning but shouldn't cause me too much trouble. I was a bit mistaken. The one thing that necessitated a significant amount of learning was the dependencies among server applications. The minimal requirement was for the implementation of a database server and team foundation server. Team foundation server requires SharePoint, SharePoint requires a correctly configured database server and if the database server is on the same server instance as SharePoint, assuming one is following best practices, it can cause permissions conflict and processor conention, if the database server is on the same instance as active directory you again run into problems. While these dependencies are readily available in documentation there isn't a complete mapping, probably due to the vast scope of the Microsoft framework as well as several past versions currently still in support.

Installing windows server 2012 r2 was as easy as installing any operating system. The trouble started when I began adding roles (Active Directory, DNS, DHCP, WINS) with Active Directory and DNS being the most troublesome. Again the reason was the learning curve; these roles are infrastructure framework systems in of themselves. Learning about DNS and how to configure it to forward to my externally hosted DNS domain, and learning about Active Directory permissions took some time.

Microsoft has done a great job of having an initial configuration of their server applications available on install. This enables developers or administrators to have a system up and running (once all dependencies are satisfied) and ready to start testing or learning immediately without also having to learn how to configure it. Microsoft also offers pre-built virtual machines with application packages already installed and configured to facilitate quicker learning and testing. I didn't go with that option as I wanted to learn the complete server setup (install/configure/administer).

5. RESULTS

After finally getting the infrastructure necessary to support software development I began the next phase of the project. I wanted to learn C#; at least from a beginners perspective. I also wanted to put into practice the methodologies that I had learned theoretically in formal education and that are in use by many companies today. Using Visual Studio and an Azure Team Foundation Instance I created a SCRUM framework project. I defined sprints with timelines and in proper agile fashion I adjusted them as I went along. Using the SCRUM framework gave me experience writing user stories, estimating work, and seeing how they translated into software, which of course wasn't that great in the beginning. I certainly accomplished my goal of learning how to use these development tools. And so upon verifying the EA was working, my user tools were properly connected to them, and that I understood how to use them in a meaningful way I began learning C# and developing my application.

Real-World Use-Case

We are entering the next phase of computing—a phase enabled by the Internet together with the concept of services, which enables the creation of powerful applications that can be used by anyone, anywhere. It increases the reach of applications and enables the continual delivery of software. In this context, software is a service—to subscribe to and use through a communication network.

.NET facilitates this idea by joining the tightly coupled, highly productive aspects of n-tier computing[2] with the loosely coupled, message-oriented concepts of the Web. Services are network-capable units of software that implement logic, manage state, communicate via messages, and are governed by policy.

[2] Also referred to as multi-tier

This fueled my decision to work within the .NET framework under C# to develop a networking library that I could implement in future applications. With server systems being separated by platform, physicality, and ownership, the best method for application communication is via messages. Messages are simply non-domain specific data packets that the receiver understands and takes action upon. An example of this is controlling a home-cinema pc. Instead of connecting directly to it you have a device connected to Wi-Fi and broadcast a UDP message on the network, any clients listening will ignore the package with the exception of the intended server. These messages can tell the server to stop/play/pause a specific movie or could be used to request a list of available titles. This is a simple consumer-based usage, however, it is upon this premise that I set out building an asynchronous networking platform upon which to implement a UDP-based client/server chat application(s).

I chose a chat program because it visually demonstrates the passing of messages from client to client or client to server. It is easy to tell whether it is passing messages correctly and responding correctly to received messages. The applications were designed to be minimal as they were designed to demonstrate and test the networking framework I had developed.

I ended up going through three iterations of the framework. The first was simply a test to see if I could make it happen, very simple user interface and simpleton classes. After that I worked on a new UI for a time but realized that I was spending a lot of time on it and doing very little coding. I scrapped that version and started on the framework as a console application. Continuing on the third implementation had a minimal UI but not barren along with a well-formed library upon which to implement the client and server.

6. CONCLUSIONS AND FUTURE WORK

Moving forward with this I intend on furthering my C# skills to continue development on a client application. Given the nature of C# it isn't best for high-performance/high-concurrency situations such as in a server I am going to recreate my server in Visual C++. These are excellent choices for further learning as they are number two and three on the top desired skills list. Additionally Microsoft's next iteration of its application ecosystem that will begin release sometime next year is set to unify all of its platforms under common base along with Android and iOS. Being able to create a solution that has common code that works on all platforms (Windows Desktop, Windows Tablet Experience, Web, Windows RT, Windows Phone, Android, and iOS) will reduce effort required significantly. This can be done using C#, C++ or even F# (which is gaining popularity) is a leap forward in simplifying development of cross-platform connected applications.

7. WORKS CITED

ANSI/IEEE. (2001). *Recommended Practice for Architecture Description of Software-Intensive Systems.* American National Standards Institute. Retrieved December 5, 2014

CyberCoders Team. (2013, November 25). *10 Most In-Demand Software Development Skills for 2014*. Retrieved December 10, 2014, from CyberCoders: http://www.cybercoders.com/insights/10-most-in-demand-software-development-skills-for-2014/

Gartner. (2013). *Enterprise Architecture (EA)*. Retrieved Dec 5, 2014, from http://www.gartner.com/it-glossary/enterprise-architecture-ea/

Platt, M. (2002, July). *Microsoft Architecture Overview*. Retrieved December 5, 2014, from Microsoft Developer Network (MSDN): http://msdn.microsoft.com/en-us/library/ms978007.aspx

Schekkerman, J. (2011). *Enterprise Architecture Tool Selection Guide.* Institute For Enterprise Architecture Developments. Retrieved December 10, 2014, from http://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&cad=rja&uact=8&ved=0CDsQFjAE&url=http%3A%2F%2Fwww.enterprise-architecture.info%2FImages%2FEA%2520Tools%2FEnterprise%2520Architecture%2520Tool%2520Selection%2520Guide%2520v6.3.pdf&ei=N0GIVOHsBs

8. APPENDICES AND ANNEXURES

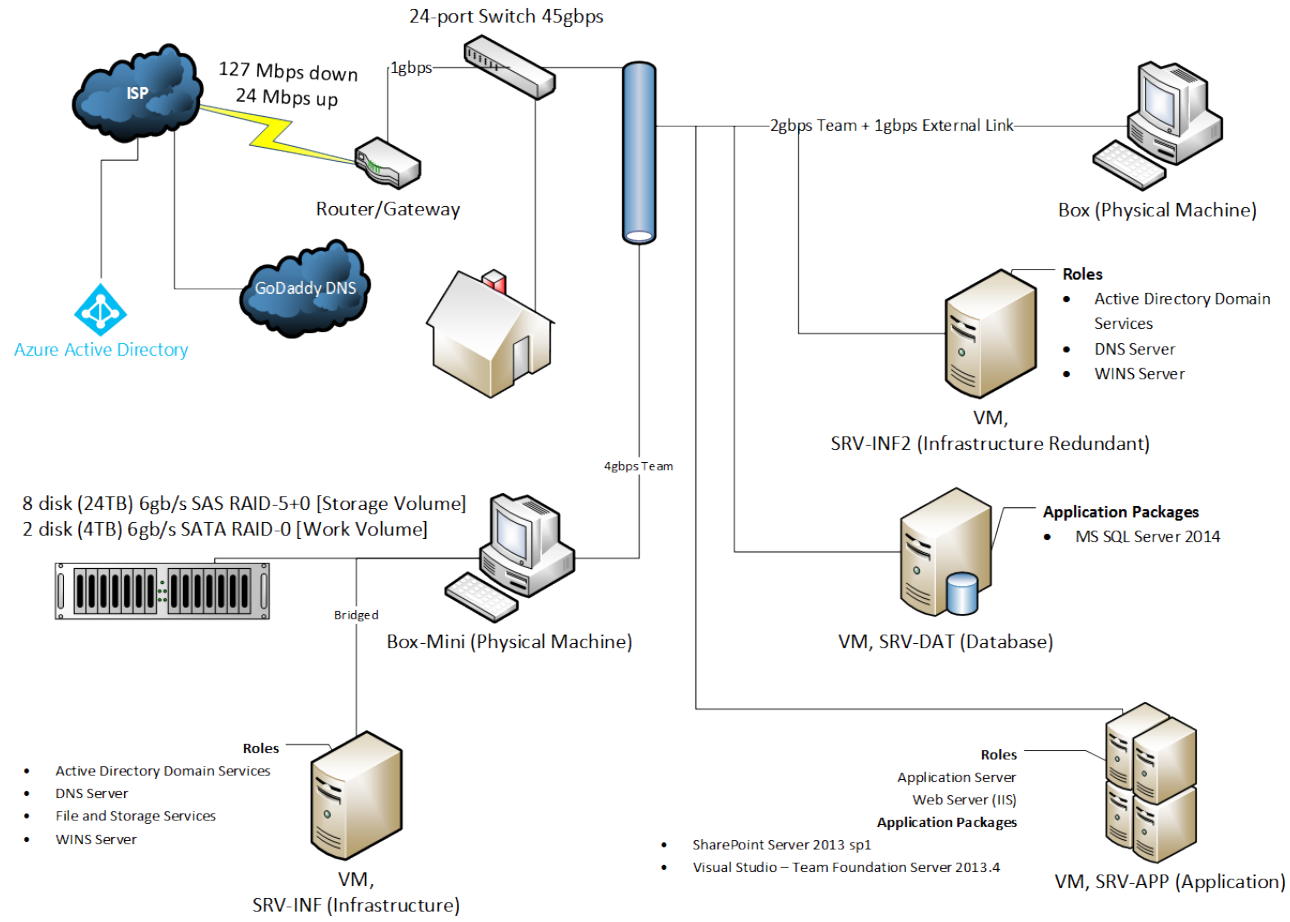

Figure 2 - Networking Topology