Spring 2021

Towards Standardizing System Design and Automating Deployment

Prerana Shekhar
San Jose State University

TOWARDS STANDARDIZING SYSTEM DESIGN AND AUTOMATING DEPLOYMENT

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Prerana Shekhar

May 2021

The Designated Thesis Committee Approves the Thesis Titled

TOWARDS STANDARDIZING SYSTEM DESIGN AND AUTOMATING DEPLOYMENT

by

Prerana Shekhar

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOSÉ STATE UNIVERSITY

May 2021

Mahima Agumbe Suresh, Ph.D.	Department of Computer Engineering
Jorjeta Jetcheva, Ph.D.	Department of Computer Engineering
Carlos Rojas, Ph.D.	Department of Computer Engineering

ABSTRACT

TOWARDS STANDARDIZING SYSTEM DESIGN AND AUTOMATING DEPLOYMENT

by Prerana Shekhar

System design is the foundation for software systems. A well-engineered and well-thought-out system design leads to the development of a software system that is scalable, reliable, extendable, and secure. It is important to be able to create a good system design. However, current ways of creating system design are inadequate. Software engineers resort to using tools like Microsoft Word, Lucidchart, Draw io, and so on to create system designs. These tools provide a basic drawing interface without the support for a standard set of system design components. There is a lack of common language for creating system designs. With the current ways of creating system design, sharing and working collaboratively throughout the life cycle of the software development is impractical. A good system design should remain faithful to the deployed infrastructure resources. To resolve these shortcomings, we propose a solution to lay the groundwork for establishing the standard. To achieve this, we abstract out the components and aim to provide a generalized view that acts as a common vocabulary to all the stakeholders. We present a prototype implementation tool that has a standard, enables version control, collaborative sharing, and automatic deployment onto multi-cloud infrastructure. The tool also provides the ability to create a configuration file such as a YAML or JSON which is a code equivalent of the system design. Furthermore, we also strive to provide a guided process that helps the beginners such as students to create a scalable, extendable, and reliable system design. The system design is then translated into a deployable entity that allocates the infrastructure resources. These features help the stakeholders to create a system design backed by a standard and understand the evolution of system design, performance evaluation, cost estimation, multi-cloud deployment, and ease of maintenance of infrastructure resources.

ACKNOWLEDGMENTS

First and foremost I would like to thank my advisor, Dr. Mahima Agumbe Suresh for her immense guidance, patience, support, and encouragement throughout the research. Without her guidance, this thesis would not have been possible.

I would like to express my gratitude to Professor Gopinath k Vinodh for taking the time to provide constructive feedback. I would like to thank Dr. Jorjeta Jetcheva for providing valuable feedback and a new perspective about the research. I would also like to thank Dr. Anamika Megwalu for helping me in editing and proofreading. My sincere thanks to the thesis committee members Dr. Jorjeta Jetcheva and Dr. Carlos Rojas for taking the time to be on the committee and providing valuable feedback.

I would also like to thank all the participants of the survey for taking the time to explain their thoughts about the research. I would like to thank all my friends for their love and support. Special thanks to my beloved friend Deepthi Jallepalli for her continuous support and motivation throughout the journey. My sincere thanks to my husband Vinay R Nagar for being the greatest support I could have wished for during this difficult year.

Last, but not least, I would like to thank my parents A V Somashekhar and Sudha Somashekhar, my sister Pooja Shekhar, my beloved niece Saanika and the entire family for their unconditional love, support, and encouragement. Without this, I wouldn't have been able to complete this thesis or pursue my dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AWS	Amazon Web Services
GCP	Google Cloud Platform
CI-CD	Continuous Integration - Continuous Deployment
EC2	Elastic Cloud Compute
HTTP	Hypertext Transfer Protocol

1 INTRODUCTION

Software systems have become an integral part of our working and personal lives. Developing reliable, stable, and safe software systems is of utmost importance. System design plays a significant role in designing and developing good software systems. It is the foundation for any software system development. It is analogous to designing the architecture of a building. According to "International Standard - Systems and software engineering – Vocabulary" [1], system design is defined as "a process of defining the hardware and software architecture, components, modules, interfaces and data for a system to satisfy specified requirements."

Despite its importance, current approaches to creating system design are not efficient. They are not backed by a standard, making them inefficient in collaborative environments. Lack of a standard results in designers resorting to drawing tools such as Lucidchart, Draw.io, Microsoft Word and the like, which are not meant to be used to create system designs. These tools often export system design as an image, which makes it difficult to track changes or versions of the design. This leads to various concerns such as lack of version control, collaborative contribution, and maintaining the history of the system design. This creates a gap in understanding the evolution of a system design and the contributions of the collaborators. Furthermore, system designs are often realized when they are deployed onto a hardware infrastructure. There is a need to understand this end-to-end process from design to deployment seamlessly. While cloud service providers like Amazon Web Services and Google Cloud Platform do a great job of providing infrastructure, they do not provide the necessary tools to create system designs. Tools like AWS CloudFormation have a steep learning curve in understanding infrastructure components that are very specific to the cloud provider. This specificity limits their use in multi-cloud system designs.

Larger organizations can afford to have a dedicated infrastructure team that works towards deploying and maintaining infrastructure. However, this demands a lot of resources. Beginners, such as students, refrain from deploying the application on cloud infrastructure due to a steep learning curve. Given the drawbacks of current processes and tools used to create system designs, there is a need to understand system designs in modern environments and propose solutions to address them. To bridge the gap of deployment on cloud infrastructure, we propose a way to automate deployment by translating the system design into a deployable entity. The prototype tool enables the user to create a system design that is backed by a standard, version control, collaborative sharing, and automatic deployment onto multi-cloud infrastructure with minimal knowledge of specific infrastructure resources and thus encouraging beginners to explore more.

The proposed solution can primarily be beneficial for students who are learning to design and develop system design. It encourages the students to create a system design based on a standard, modify it along the process of learning, visualize the evolution of the system design, enable version control and automate deployment on cloud infrastructure. The system design can be deployed on various cloud services such as AWS, GCP, Azure, and private cloud, thereby helping the users to understand and explore the equivalent infrastructure components. Hence, the proposed prototype tool can be utilized as a guided educational tool to motivate the students by bridging the gap.

1.1 Research Objectives

The first objective of the research is to study and understand the current definitions and processes of creating system designs. This also involves studying existing tools and identifying features that assist in system design or lack thereof.

The second objective is to propose solutions to the problems identified. This involves laying the groundwork for a new standard for system designs while considering

extensibility. The solutions should also address requirements around deployment, version control, shareability, and collaboration.

The third objective of the research is to implement a prototype tool based on the proposed solutions to test the end-to-end process of creating system designs for a limited set of features that demonstrate the feasibility of the solutions.

1.2 Thesis Organization

The rest of the paper is organized as shown in Fig. 1.

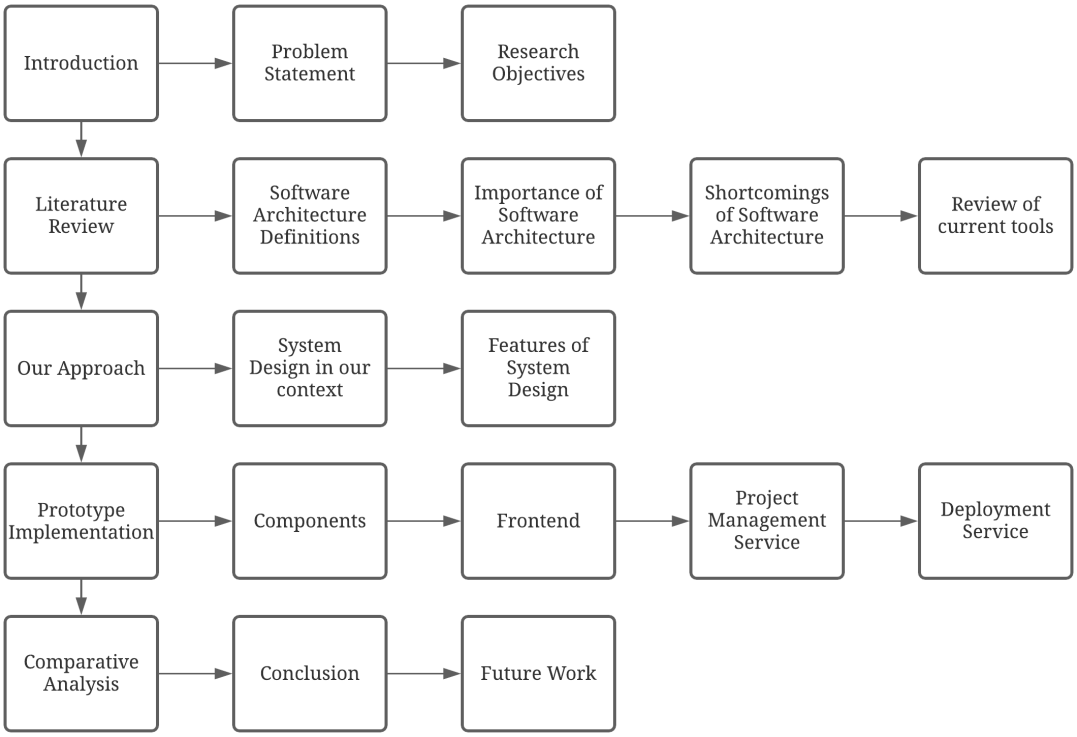

Fig. 1. Thesis organization.

2 LITERATURE REVIEW

We first aim to identify the various definitions of software architecture and the analogy between classical building architecture and software architecture. We then highlight the importance and need for the software architecture. Further, we analyze the shortcomings of the current tools that are used to create the system design.

2.1 Software Architecture Definitions

This section aims to describe the definitions of software architecture as explained in various papers.

2.1.1 Perry and Wolf

One of the pioneer papers [2] made the first attempt to define the software architecture [3]. In this paper, Perry and Wolf define the software architecture as a model that consists of three main components:

1) Elements
2) Form
3) Rationale

Furthermore, they classify "elements" into three different groups: "processing elements, data elements and connecting elements" [2]. The processing elements are the components that transform the data elements. The data elements contain the information that undergoes transformation and connecting elements are components that help in holding the entire architecture together [2]. Perry and Wolf state that the "form" comprises of "weighted properties and relationships" [3]. The form indicates the relationship between the elements. They explain "rationale" as the "system constraints that form the basis for the architecture" [3]. It indicates the motivation behind the decisions and the choice of elements.

Perry and Wolf [2] highlight the importance of software architecture. They explain that the primary purpose of the architecture is to enable automated document analysis and

reveal the various problems that could have gone unnoticed. They also describe two primary types of analyses, namely, consistency analysis and dependency analysis [2]. Consistency can be seen in various aspects of the architecture such as consistency within the architecture and its styles, consistency with the requirements, and the design. Dependency analysis evaluates the inter-dependency among the requirements, architecture, and design.

2.1.2 Bass et al.

Bass et al. define [4] software architecture as "a structure of a system that comprises of system components, their properties and the relationship between them." They bring about a contradicting analysis of the system design being abstract as well as specific. They convey that the architecture must abstract away some information from the design yet provide enough information to represent the requirements [4], the relationship between the components.

2.1.3 Kruchten

Kruchten [5] presented the idea of software architecture by defining the following four terms, namely, "logical view, physical view, process view and development view" [2]. The logical view defines breaking down the system into logical sections using the given requirements. The process view describes how the functional requirements match with the process. The physical view [5] represents the translation of the software components onto the hardware components. The development view focuses on the actual software development environment [5].

2.1.4 D. Garlan and M. Shaw

D. Garlan and M. Shaw defined software architecture as a "hierarchy of software design" [6]. They explain that the software architecture consists of "components, connectors, ports, and roles." They explain that the "component" can be a "class, function,

procedure, module, process or a group of small systems" [6]. It can represent a function or data storage. They also explain that the larger the component, the higher the number of features included in the component that leads to good quality systems with lower errors [6]. The "connector" acts as a bridge between the components. A connector is independent of the application. They connect various components and transfer data and information between the components [6]. An example of a simple filter is a pipe-filter whereas the client-server protocol is a connection between the database and the application is a complex connection [6]. "Port" is described as an interactive method between the components and the application environment. Every component can have one or more ports. The component can represent many different interfaces through various ports connected to them [6]. The "role" indicates the connection between the connectors and the outside environment [6]. Each connector has at least two roles, for receiving information and sending information. Hence it's called a dual connector [6]. It can also have multiple roles like broadcasting events and any number of roles for receiving event [6]. Hence, it is called a multi-connector.

2.1.5 N. Medvidovic and R. N. Taylor

Medvidovic and R. N. Taylor [7] explain software architecture as "the set of principal design decisions made during its development and subsequent evolution." They state that to develop a high quality product, a good design decision plays an important role [7]. The quality of software systems is proportional to the system design quality. It is very rare to notice a good software system with poor design. The paper defines that a software system consists of structural elements: components, connectors, and configurations [7]. They define components as "units of computation in a system," connectors as "loci of the interactions between software components" and configurations as "arrangements of software components and connectors, and the rules that guide their composition" [7]. The authors further explain that the principal design decisions consist of mapping of these

software components onto hardware components of deployment, by considering reliability, scalability, security, and efficiency [7].

2.2 Analogy Between Classical Architecture and Software Architecture

Perry and Wolf [2] bring up an analogy between the classical field of building architecture and software architecture. Although classical building architecture and software architecture are highly distinct, they have multiple similar points of view [2]. They highlight the following four aspects that hold good for both building architecture and software architecture: “multiple views, architectural styles, style and engineering, style and materials” [2]. They explain the relevance of “multiple views” in building architecture as floor plans, exterior views, top-down views [2] that are helpful to provide explicit design details to the builder. Such a multi-view system provides specific details about electrical, plumbing, heating, and air-conditioning [2] designs to be considered by the builders. Similarly, in software architecture, multiple views help in thinking about the architecture from a design and implementation perspective. The authors state that the relationship between the architectural styles and engineering principles is of extreme importance [2]. The look and feel of the building are closely related to the right balance between the architectural style and engineering principles. The architectural style emphasizes the design constraints, design decisions, and the relationship between the elements [2]. Also, the connection between the materials and the style of the buildings are comparable. Perry and Wolf state that “one does not build a skyscraper with wooden posts and beams” [2]. On a similar note, the software architecture needs to be implemented on the right set of components based on the load distribution and the application requirements.

N. Medvidovic and R. N. Taylor [7] also highlight the analogy between the construction building and software systems. Software architecture should be at the heart [7] of a software design. They emphasize that the design should be given “higher

weightage than process, analysis, and programming" [7]. As designing a good architecture of a building helps in the stability, durability, and strength of the building, similarly, a well-thought-out software system design would last a lifetime. It helps in the evolution of the software system. They also stress focusing on the architecture over its entire lifespan instead of a limited-term [7]. Furthermore, they explain that having a good design does not guarantee that a good building will be constructed [7] as there will be some gaps and modifications during the course of construction. Similarly, even in software systems, there is no validity that the system would be faithful to the design developed due to the shortcomings along the process [7]. Hence, it is important to be able to ensure that the implemented system remains faithful to its intended system design.

2.3 Importance of Software Architecture

The three fundamental reasons stated by Clement [8], to indicate why software architecture is important is as follows:

1) "Mutual communication": Software architecture is a high-level representation of the deliverable that helps most of them understand the requirement by creating a common basis [8] of communication.
2) "Early design decision": The early design decisions help in keeping the components connected and these decisions [8] weigh out far more than the individual decisions made during the development. These decisions also help in the deployment phase and maintenance of the system.
3) "Transferable abstraction of a system": Software architecture has to be abstracted out so that these components and subsystems can be used in different applications [8] allowing reuse of the system architecture.

2.3.1 Architecture is the Common Ground for Communication

Multiple stakeholders refer to the architecture with various perspectives. The manager looks at the architecture with the perspective of how independently the team can work on

the system, whether the deliverables can be implemented on schedule, and cost estimation [8]. The programmer looks at the system architecture to work through the strategies to develop the requirement. Hence, the software architecture acts as a common ground to communicate the various aspects of the system with different perspectives. It helps in understanding the complexities of a large system with ease.

2.3.2 Architecture Provides the Developers a Set of Guidelines

An architecture provides a set of structural design decisions for the implementation. It defines the set of components and the relationship between these components that help throughout the life cycle of the system [8]. This also helps to maintain the project.

2.3.3 Architecture Helps Manage Changes

Studies indicated that 80% [8] of the software system's cost is involved after the deployment of the project. Oftentimes we need to modify the software systems during their lifetime. They need to enhance the system due to improvement in technology. Refactoring of systems is necessary to increase efficiency, reliability, scalability of systems [8]. It is highly important to address these changes to understand the impact of the changes on the existing system, the impact of the changes on the relationship between the components and the behavioral aspects of the system [8]. Clement classifies the changes as "local, non-local and architectural changes" [8]. A local change is referred to as the changes in the component. A non-local change refers to the changes in multiple components. Architectural changes impact the way the system is designed and the way the components communicate with each other.

2.4 Shortcomings of Software Architecture Definitions

Although there are numerous definitions of software architecture, concerns still exist about the formal definition because of the nature of various parameters to be considered, which concepts to be included, and which concepts to be excluded [3], [8]. Perry and wolf

summarize the shortcomings of software architecture based on Clements description in "Software Architecture, An executive Overview" [8] as below.

1) "Advocates bring their methodological biases with them." As seen from the various definitions, they mostly agree at the core however differ at the fringes [8]. Some of them state the importance of process while others emphasize on the functionality to be allocated to the components. Hence, it is important to understand the meaning of the definition in the given context.
2) "The study of software architecture is constantly evolving." People refer to the previous definitions and modify them according to their needs and broaden the meaning of software architecture [8]. Hence, a formal definition has not been converged yet.
3) "Although the research is ongoing, the field is pretty new."
4) "The most common terminologies are ambiguous and do not have a clear definition." The concept of a "component" is highly overloaded. Oftentimes there is no clarity of what goes inside a component. Clement refers to a top-level architecture [8] and indicates that there is no proper understanding of these layers. The architecture also fails to explain the importance of the "links" connecting the various components. It also does not indicate the direction of data flow and there is a high level of information missing [8]. Furthermore, Clement also studies layered architecture. He explains that these layers indicate hierarchical structure between them. However, it fails to highlight the concept of a component, and a close study of the architecture reveals that there is a high level of overlap between the layers.
5) "The terminologies are over utilized" [3], [8] and modified according to one's needs and hence the meaning of these terminologies is getting diluted in software architecture.

2.5 Current Tools Used to Create System Design

To create system designs, we use several tools like Microsoft Word, Lucidchart, Draw.io, and so on. We study these tools to understand the advantages and disadvantages of these tools, in the field of system design.

2.5.1 Lucidchart

Lucidchart is a web-based diagramming software that is used to create flowcharts, organizational charts, website wireframes, mind maps, software prototypes, and many other diagram types [9]. Lucidchart started in December 2008 based out of salt lake city, Utah. Currently, Lucidchart is used in over 180 countries by more than 15 million users, from sales managers mapping out prospective organizations to IT directors visualizing their network infrastructure [10]. The primary purpose of using Lucidchart is to create diagrams by utilizing the various shapes, designs, images from the component registry. Lucidchart also allows the user to start from a specific diagram template. It is highly focused on creating diagrams.

2.5.1.1 Advantages.

1) Primary purpose is to create diagrams such as flowcharts, organization charts, UI workflows, developing business strategies, and so on.
2) It also allows the users to share and work collaboratively on the ongoing project.
3) It allows sharing of an image representation of the diagram on several integrated platforms like Jira, Confluence, and Slack.
4) Lucidchart also allows the user to integrate with cloud services and create a graphical representation of the deployed infrastructure.

2.5.1.2 Disadvantages.

1) Lucidchart can only be shared as an image and not a live document.
2) It is primarily a drawing tool and does not cater to the needs of a system design development. Every user can take the liberty to represent the components of the

system design in a way that is convenient to them. A component can be represented either as a square or a rectangle. This dilutes the concept of having a standard.

3) A code equivalent representation of the diagram is non-existent.
4) It does not maintain a history of the system design. Hence, it fails to capture the evolution of a system design throughout the process of decision-making and the life cycle of the application.

2.5.2 Draw.io

Draw.io or recently known as diagrams.net is another popular tool that is widely used for creating diagrams. It is a free, high-quality drawing tool. Draw.io was founded in 2000 and its headquarters is located in Northampton, Northamptonshire, UK [11]. It allows the users to create various types of diagrams such as flowcharts, entity-relationship diagrams, building floor plans, electrical diagrams [12] and much more. It provides the users with 3 main components - shapes, connectors, labels [12].

2.5.2.1 Advantages.

1) Draw.io allows the users to create various types of diagrams.
2) It allows the users to share diagrams via google doc or One drive files. It also provides the users to integrate with other tools like Atlassian Confluence Cloud, GitHub [13].

2.5.2.2 Disadvantages.

1) As Draw.io is a generic drawing tool, it does not account for the standardization of system design components.
2) Draw.io allows the users to share files via Google doc or One drive files [14]. However, sharing a system design via these files is not efficient as it is shared in the form of an image. It defies the ability of the user to be able to keep track of the changes made by multiple users. Also, it does not support collaborative work.
3) Keeping track of different versions is not possible.

4) It is not a deployable entity.
5) It does not estimate the performance of the system and the cost involved when deployed on cloud infrastructure.

2.5.3 Microsoft Word

The first version of Microsoft Word was released in 1983, primarily as a text editor, and developed further to support various fonts, images, WordArt, undo, redo and many more features [15]. Microsoft Word provides a plain slate to draw diagrams using various standard shapes, arrows, and images.

2.5.3.1 Advantages.

1) Microsoft Word is very popular and widely used.
2) Integration of Microsoft Word with other tools is seamless.
3) It can be shared and allows for collaboration.

2.5.3.2 Disadvantages.

1) Microsoft Word is not meant for creating system designs as it does not provide standard system design components.
2) Although it can be shared, it does not keep track of the changes made by different people in a true sense. Hence, it does not support collaboration and version control of the created system design.
3) It does not convert the created system design into a deployable entity.
4) It does not take into account the performance, security, and cost involved when the application is deployed on cloud infrastructure.

2.5.4 AWS CloudFormation

AWS CloudFormation is a service provided by Amazon Web Services to create infrastructure diagrams for the application [16]. The user can create an infrastructure diagram either by choosing a set of AWS services from the resources registry or by using

a JSON/YAML configuration file. The user can save a version of the template on AWS S3. Hence it provides version control to the user. Once the template is ready, the user can provision AWS resources.

2.5.4.1 Advantages.

1) AWS CloudFormation allows the user to create infrastructure templates by utilizing aws resources. Both graphical, as well as infrastructure as a code representation, are available.
2) It helps the users to estimate the cost involved for the resources created [17].
3) It allows sharing and editing the template with the collaborators by keeping track of the changes made to the system design.
4) The user can provision and maintain the resources using the template created.

2.5.4.2 Disadvantages.

1) AWS CloudFormation does not provide an abstraction of the components deployed. It is very closely tied to the AWS cloud infrastructure. It is highly necessary to have a good understanding of the AWS infrastructure resources, the various type of resources, specific components, their configurations, and so on. Hence the learning curve is pretty high.
2) The graphical representation of the infrastructure resources corresponds purely to the deployed infrastructure. It is an infrastructure diagram and not a system design.
3) It does not integrate with other cloud providers. Hence, the users are mandated to use AWS infrastructure.

2.5.5 Git

Git is free and open-source software that is used to track changes of any file and help software developers to work collaboratively [18]. Git aims to provide speed, data integrity, parallel workflows, version control, and collaboration. Git helps the programmers to have a complete history of the code developed and version tracking of all the changes made

along the lifecycle of development of the project [18]. It helps to roll back to an earlier version in case of failure and protects the application from failing in the production environment. It also helps in parallel workflows by enabling the users to create multiple branches. The following are the advantages for system design.

1) Git provides a history of all the changes made to the code.
2) Git keeps track of all the changes made by collaborators.
3) Git helps to roll back to a previous version.
4) Git helps to maintain private and public repositories.
5) Git provides integration with several services to set up CI-CD pipeline and enables to automatically push the changes to production.
6) It also helps in maintaining different environments such as development, staging, and production. Therefore, it maintains different versions of code in different environments.

2.5.6 Inference from Literature Review

From the literature survey, it is clear that a system design is a highly overloaded term. It is used in several different perspectives. There is no standard way of defining or representing a system design. System design created using Lucidchart, Draw.io, and Microsoft Word files is not the ideal way of creating it. They have numerous challenges and do not cater to the needs of creating a system design.

One of the drawbacks of current ways of creating system design is not having the history of the system design. This creates a gap in understanding the progression of the system design throughout the lifecycle of the application. Having a good understanding of the various decisions and modifications made during the journey of a system design is vital. This helps to overcome the flawed decisions and incorporate the right set of decisions for future work.

Oftentimes, the deployed infrastructure does not remain faithful to the system design created due to the changes during the development of the application. Hence, this creates a gap between the system design and the deployed infrastructure resources. Therefore, to overcome all the above-mentioned challenges, it is important to have a system that encourages the creation of a system design based on a standard that has the capability of version control, collaboration, and automated deployment. In the further sections, the paper strives to propose an approach and a prototype implementation to resolve these issues.

3 OUR APPROACH TO STANDARDIZING SYSTEM DESIGN AND AUTOMATING SOFTWARE DEPLOYMENT

As described in the literature review, there are several shortcomings of the current ways of creating system design. There is a need to develop a standard for system design, paving the way for a more streamlined and portable design process and automated deployment. We achieve this goal by providing the first steps towards achieving a standard and demonstrate it with a prototype implementation.

First, to identify the gaps in current tools as seen by users, a survey was conducted to determine if these shortcomings are currently prevalent to this day. In this section, we first provide the results of the survey conducted and then introduce a set of concepts in the context of the research. Furthermore, we strive to propose an approach to overcome the problems stated in the literature survey.

3.1 Survey

We surveyed to validate the shortcomings of the system design on 65 individuals. Below are the details of the participants and their responses about the current ways of creating system design.

3.1.1 Participant Information

1) The survey was conducted on 65 individuals of which 63% were master's students in the field of software engineering, 34% were software engineers and the rest included professors, data scientists and hardware design engineers.
2) The participants' experience in the field of software engineering varied from less than a year to up to 25+ years. Fig. 2 shows the percentage of participants with a range of work experience in the field of software engineering

3.1.2 Participants Response

1) Among the participants, 98.5% of the individuals stated that they create a system design before developing an application.

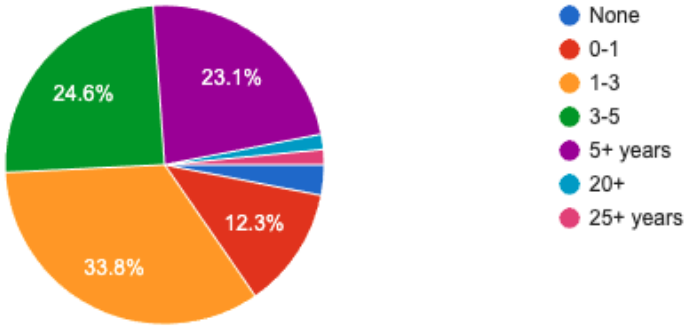

Fig. 2. Percentage of participants with a range of work experience in the field of software engineering.

2) To create a system design, 55% use Draw.io, 37% use LucidChart, 24% use Microsoft Word Document, and 21% use various miscellaneous tools like Visio, Microsoft PowerPoint Presentation, Quip etc. Fig. 3 shows the percentage of users using various tools for system design.

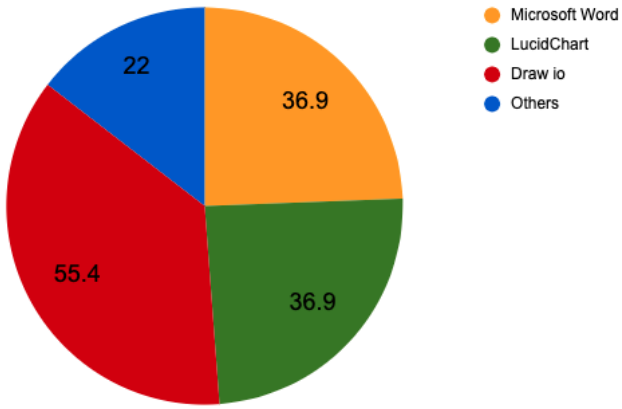

Fig. 3. Percentage of users using various system design tools.

3) Among the participants, 70% said that they prefer creating a system design from an existing template and 30% said that they prefer to create from the beginning without a template. Fig. 4 shows the percentage of users starting the system design from a template or without a template.

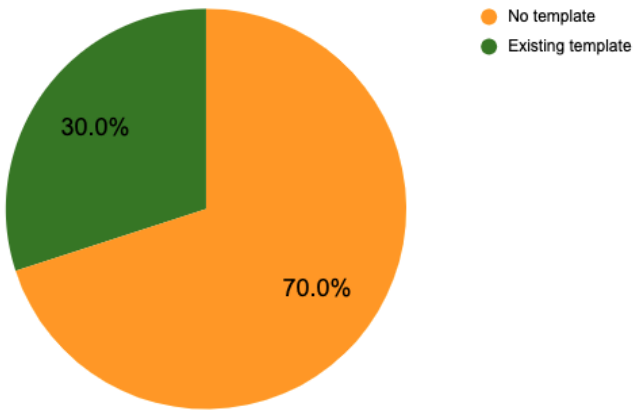

Fig. 4. Percentage of users using existing template and no template.

4) Among the participants, 90% stated that they share the system design in an image format and the rest shared the document links, for example, sharing PDF files, Microsoft Word files, PowerPoint presentations, Visio links, Lucidchart links. Fig. 5 shows the percentage of users sharing the system design in the format of an image.

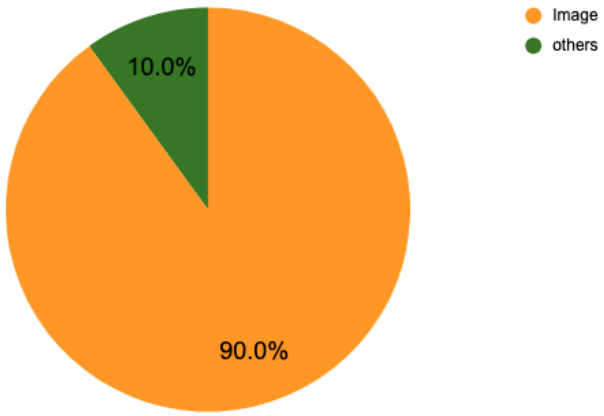

Fig. 5. Percentage of users sharing system design as a image.

5) Among the participants 93% of the participants said that it would be helpful to have a tool that allows sharing, helps to work collaboratively and enables version control.
6) Given a choice to draw a system design from the graphical components or to create it using a configuration file, 75.4% of the participants preferred to draw and 21.5%

preferred to use the configuration file and the rest opted for both. Fig. 6 shows the percentage of users who prefer to create the system design using visual representation of components and configuration file.

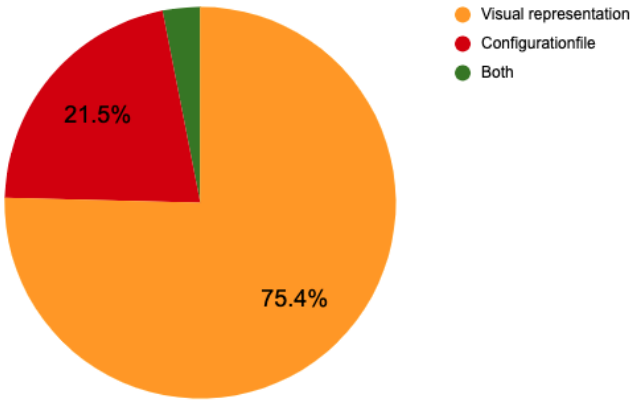

Fig. 6. Percentage of users who prefer to create the system using visual representation components and configuration file.

7) All the participants agreed that it is important to have a good system design that helps in achieving the end goal.

3.2 System Design in Our Context

From the literature survey, we understand that there are several shortcomings of the current ways of creating system design. To address these problems, we first introduce a set of concepts and their meanings in the context of this research. The below section aims to introduce these concepts.

3.2.1 Our Definition of System Design

A system design is a graphical or configuration representation of the standard components, their relationships, and dependencies, which can be translated into a deployable entity by mapping them to the infrastructure resources. A system design is expected to be secure, collaborative, based on a standard, allowing version control,

cost-efficient, provide high performance, and easily deployable. The terms used in this definition are explained in the below sections.

3.2.2 Component

A “component” is an entity that represents an infrastructure resource. A component is independent or dependent on another component. A component has relationships with other components. It is an abstract representation of the equivalent resources provided by different cloud service providers.

3.2.2.1 Types of Components. A database server, web server, Kubernetes cluster, load balancer, pub-sub service, queue component, API gateway are some of the examples of a component. These components are configurable and the details of the “configuration” are defined in the further sections.

3.2.2.2 Properties of Components. A component has the following properties:

1) Name: Indicates the name of the component.
2) Description: Describes the component.
3) Component Type: Indicates the type of the component. For example, a “server” component, a “database” component.

3.2.3 Component Registry

A “component registry” is a collection of components that are used to create a system design. This paper attempts to define a set of standard components that are mentioned in the above section. Every component in the component registry has a description that indicates the purpose of the component.

3.2.4 Connection

A “connection” is an entity that connects various components. It indicates the data flow between the components or the dependency between the components. It is represented by an arrow that can be unidirectional or bidirectional.

3.2.4.1 Types of Connections. As part of this research, we define 2 main types of connections between the components.

1) Data flow connections: This indicates that there is the data flow between the components and also indicates the direction of data flow among the components.
2) Dependency connections: This indicates the dependency between the various components. This type of dependency indicates a hard dependency on another component. They are functional and sub-component dependencies. For example, a connection between a load balancer and its dependent servers.

3.2.4.2 Validations. The connections between the components are validated to verify if it represents a logical connection among the components. The direction of the connection is also verified to represent a valid data flow or a valid dependency between the components. For instance, a server cannot be connected in front of a load balancer. It defies the meaning of the load balancer component.

3.2.5 Configuration

A "configuration" is a setting of the component that acts as a set of rules or guiding principles. It specifies the details of a component. Every component has a default setting and a custom setting. Based on the most appropriate configuration possible, the components are provided a set of default configurations. It also provides the ability to modify the default configuration according to the needs of the application. The configuration can be classified into two types.

3.2.5.1 Component Configuration. Component configuration indicates the specifications of a component. In a multi-layered approach, the different specifications of a component are specified in multiple stages. The concept of multi-layered design is explained in the below section. A configuration is a setting of the input and output ports for the servers, specifying the type of database, the type of load balancer to be used, the details of a Kubernetes cluster. For example, in the first level a "database component" is

added to the system design. In the second level, the type of database such as a mongo DB, Postgresql, MySQL is configured. In the third level, the component is configured to an appropriate infrastructure resource. The component can also be configured to deploy the application on the infrastructure resources using a containerized application URL like docker.

3.2.5.2 Deployment Configuration. Every component of the system design is configured to map to an infrastructure resource. It enables the system design to be converted into a deployable entity. The configuration of the deployment represents the type of resources to be used to create an infrastructure resource. Validations are made using meta-data information to ensure that these components are connected to a logical equivalent component on the infrastructure.

3.2.6 Deployment

A "deployment" is a configuration in the last level of the multi-layer design. This provides the user to select the type of deployment service providers such as Amazon Web Services, Google Cloud Platform, Azure, and any private cloud service. This enables to choose the type of service based on several parameters like the cost involved, performance evaluation, security, reliability of the infrastructure. At this level, the system design components are mapped to the infrastructure resources based on the type of cloud service provider chosen. The deployment level is a translation of the system design components to its respective infrastructure resources. All the components of the system design are mapped to the default infrastructure resources based on metadata configuration. These default configurations can also be modified to have a set of custom configurations.

3.2.7 Multi-layer Design

A "multi-layer design" is a concept in the process of creating a system design. There are 3 main levels. The first level enables the provision of a set of components and connections. In the second level, various configurations of the components can be set. In

the third level, deployment configurations are set and the deployable system design provisions the respective infrastructure resources.

3.2.8 System Design Canvas

A “system design canvas” is a graphical representation of the prototype model that helps to create a system design by selecting the components from the component registry, creating connections, and setting the configurations for each of the components. The system design canvas is present in each layer of the prototype model.

3.2.9 System Design as Code

A “System design as Code” is a configuration file representation of the system design created by adding the components and their configurations on the system design canvas. It can be represented in 2 ways, JSON and YAML configuration files. The configuration files indicate the components of a project, their relationship among the components, dependency among the components, their configuration specifications, the type of infrastructure resources mapped to it and so on. These configuration files can be used to create a graphical representation of the system design and vice versa.

3.3 Features of System Design

In this section, we propose the following features of system design. These features have been implemented in the prototype model that is discussed in the following section.

3.3.1 Standard for System Design

From the literature survey, it is known that the current ways of creating system design lack standards. The term system design is overloaded and has multiple perspectives. Hence, It is important to create a standard for system design. Standards provide people and organizations with a basis for mutual understanding and are used as tools to facilitate communication [19]. ISO defines a standard as “A standard is a document that provides

requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes, and services are fit for their purpose" [20].

Benefits of having a standard:

1) Common language of communication and performance evaluation [20]
2) Increases security
3) Easy to maintain
4) A comprehensive look
5) Cost efficient
6) Bug rectification

As part of this research, we aim to define a standard by defining the standard components, standard configurations, a standard way of representation of system design in a configuration file.

3.3.2 Sharing

"Sharing" is an important concept in software engineering. It helps multiple collaborators to work on the same document, file, or code. It also refers to the process of having the ability to share one's work with the community. Having a reliable system design management software allows for real-time collaboration. One should be able to create a system design and share it with collaborators. It helps all the collaborators to be in sync. Furthermore, the system design can be shared by adding collaborators on the prototype model, using URL or by exporting the information to a file. The user can enable "read" or "edit" permissions to the collaborators. The collaborators of the system design are notified when there is an update on the document. The created system design can be embedded in other products like Microsoft Word document, quip, and so on thereby keeping all the documents in sync with the latest version of the document. An analysis of the shared document can bring about profitability to the organization [21].

Sharing of documents has the following benefits:

1) Common language of communication and performance evaluation [20]
2) Enhances efficiency
3) Considerable transparency
4) Gives an overview of the flow of ideas
5) Helps to determine drawbacks
6) Efficient time management

3.3.3 Version Control

"Version control" is another important feature in software engineering. It refers to the ability to track changes in a file, code, or system. In this paper, we aim to propose a solution that enables version control of the system design. The concept of version control is the same as the version control feature of git as described in the literature survey. Version control helps to keep track of all the changes made to the system design by various users. This helps to understand the evolution of the system design during its life cycle. Version control is beneficial for the following reasons [22]:

1) Collaboration
2) Storing versions properly
3) Restoring previous versions
4) Understanding what changes were made in every version
5) Backup

3.3.4 Automated Deployment

Automation is an essential part of software engineering. In this paper, we bridge the gap between the system design and software infrastructure resources. In the prototype implementation model, we see that the users create infrastructure resources from the system design.

Benefits of automated deployment:

1) One does not have to maintain the infrastructure resources manually
2) Reliability of the system can be increased with efficient monitoring tools
3) Reduces efforts and human errors
4) Faster deployment
5) Increases developer productivity and efficiency
6) Maintenance of the infrastructure is easier
7) Ability to have multiple environments - staging, production
8) Ability to deploy on multi-cloud services

3.3.5 Performance

Analyzing the performance of any system is important. In this paper, we strive to analyze the performance of the system by taking into consideration the type of application that is deployed. It is important to understand the load of the application, data involved in processing to understand the required specifications of the resources. By collecting application-specific information and evaluating the capabilities of the infrastructure resources, we can measure the performance of the system.

3.3.6 Cost

The cost of the deployment of infrastructure resources varies based on how small or large infrastructure is needed for the application. Infrastructure cost is calculated based on hourly, monthly, yearly, no of hits, data volumes, and so on. Considering these factors we can evaluate the approximate total cost involved. Since the prototype model supports multi-cloud service providers, the user can choose the most cost-efficient cloud service provider.

3.3.7 Security

Security is one of the most important concerns of any organization. It is highly important to have a secure infrastructure. By configuring the components in private subnets, adding security group configurations, the system is made secure.

3.3.8 Guided Process

One of the applications of the prototype implementation of the tool is to present it as an educational tool. This tool can be used by students who are starting to learn system design and infrastructure deployment. The students can begin by adding components to the canvas in the first level. The tooltip provided against each component helps the student to learn more about the component and its usage. The tool also provides a list of standard components and their purpose. When the student chooses a load balancer component, we guide them by indicating the need for additional servers. This educates the student about load balancer, its purpose, and the importance of having scaling and reliability in the system design.

In the second level, the student can configure additional parameters such as the type of database to be used, port numbers, type of load balancer, and so on. We guide the student by setting all the default parameters.

In the third level, the student can choose the type of cloud infrastructure elements. All the standard components are mapped to their respective default infrastructure resources. Once the student clicks on the deploy job, the infrastructure resources get created on the cloud account, whose credentials are set by the user. Thereby it solves for the high learning curve. The entire process acts as a guided process to encourage the students to learn more.

4 PROTOTYPE IMPLEMENTATION

Fig. 7 shows the overall system architecture of the prototype implementation.

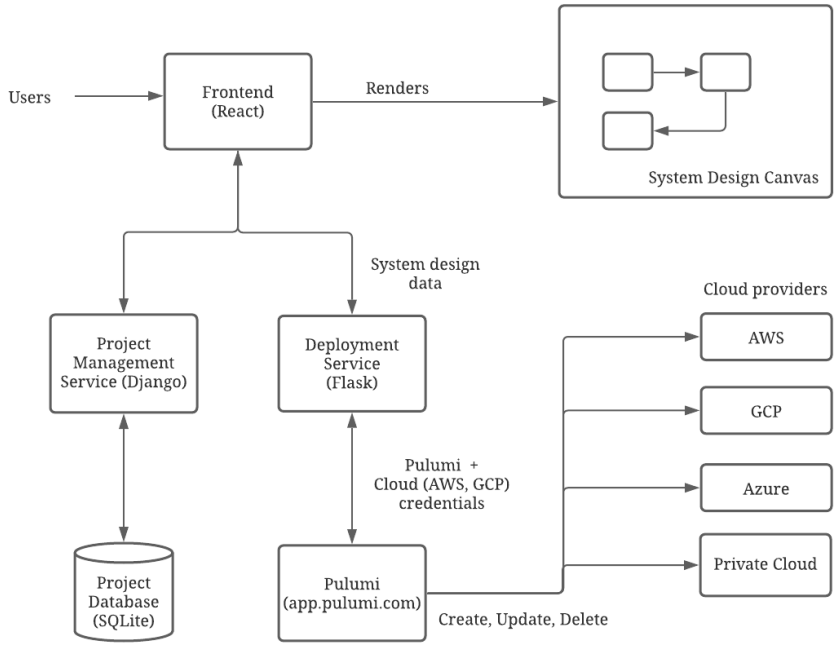

Fig. 7. Prototype system architecture.

The following sections describe the main concepts of the prototype architecture.

4.1 Components

Components are the fundamental units used in system design. A component roughly has a mapping to an actual infrastructure component when deployed to the cloud. Every component is self-represented and provides a clear description of its purpose and usage. It also has a type that mainly indicates its function. For the prototype, the following component types are supported.

1) Server: representation of a machine that can satisfy HTTP requests on public and private clouds

2) Database: representation of a database management system. For prototype, only relational databases are supported
3) Load balancer: representation of software or hardware used to automatically distribute incoming traffic across multiple targets, such as server instances in one or more geographically distributed areas
4) Kubernetes cluster: representation of a set of nodes that run containerized applications

The components are represented in the database in a table called, **Components** through foreign key relationships to table **ComponentTypes** as shown in Table 1 and Table 2, respectively.

Table 1
ComponentTypes

Column	**Column Type**
id	integer
name	text
created_at	datetime
updated_at	datetime
deleted_at	datetime
deleted_why	text

4.2 Frontend

This is the frontend application the user interacts to perform all user activities.

The application is built using React [23] which is an open-source, frontend, JavaScript library for building user interfaces. The main features of the frontend application are listed below.

Table 2
Components

Column	Column Type
name	type
id	integer
name	text
level	integer
type_id	integer foreign key references ComponentTypes(id)
image	text
description	text
created_at	datetime
updated_at	datetime
deleted_at	datetime
deleted_why	text

1) Projects: Users can create projects to maintain their system designs. Every system design will be associated with one project and one project contains only one system design. This is the highest-level entity the user interacts with.
2) Layered Visual Canvas: The main interacting entity to create system designs. It is a layered visual representation guiding the user from an abstract representation of the system to an exact representation of the cloud infrastructure.
3) Project Commits: User saved changes are called commits. Every commit is accompanied by a message that represents the state of the project at that time.
4) Deployment: Projects can be deployed onto cloud infrastructure to create actual infrastructure components. The prototype supports AWS and GCP platforms. One project could be deployed to both platforms and any number of times.

5) YAML processor: System designs can be created from a YAML representation. A processor converts the YAML representation into a visual representation.

4.3 Project Management Service

This is the backend service for most user actions done in a project except for deployment.

The service is implemented using Django Rest Framework [24], a framework for web Restful Web APIs. It is implemented on top of Django [25] which is a Python-based free and open-source web framework that follows the model-template-views (MTV) architectural pattern. The following are the high-level concepts under the project management service.

4.3.1 Projects

A user created project that contains the system design. A project is represented in the database as show in Table 3.

Table 3
Projects

Column	**Column Type**
name	type
id	integer
name	text
description	text
is_template	bool
cloned_from	integer foreign key references Projects(id)
created_by	integer foreign key references Users(id)
created_at	datetime

4.3.2 Project Components

The standard components that are used in a project are represented as project components. A project component is a binding of a standard component to a project. Every component can contain additional user-supplied configuration parameters. This configuration is used while deploying a project component onto the cloud infrastructure. The prototype supports the following configuration parameters.

1) database_selection
2) database_tag
3) instance_tag
4) input_port_number
5) output_port_number
6) load_balancer_name
7) load_balancer_selection
8) cluster_name
9) cluster_docker_image
10) cluster_minimum_size
11) cluster_maximum_size

Project components are represented in the database as shown in Table 4.

4.3.3 Connections

Connections represent dependencies or data flow between project components. Some connectors are purely representational like between a backend and database server while certain imply actual dependencies like between a load balancer and connected backend servers.

Connections are represented in the database as shown in the Table 5.

Table 4
ProjectComponents

Column	Column Type
id	integer
standard_component_id	integer foreign key references Components(id)
project_id	integer foreign key references Projects(id)
config	json
created_at	datetime
updated_at	datetime
deleted_at	datetime
deleted_why	text

Table 5
Connections

Column	Column Type
id	integer
from_project_component_id	integer foreign key references ProjectComponents(id)
to_project_component_id	integer foreign key references ProjectComponents(id)
created_at	datetime
updated_at	datetime
deleted_at	datetime
deleted_why	text

4.3.4 Commits

Commits can be thought of as snapshots or milestones along the timeline of the project. They capture the state of the system at that particular point in time. Users can

save the project with a commit message. They can also see all versions of the project saved over a period of time.

Commits are represented in the database as shown in the Table 6.

Table 6
Commits

Column	Column Type
id	integer
config	json
committed_at	datetime
commit_message	text
project_id	integer foreign key references Projects(id)
committed_by	integer foreign key references Users(id)

4.3.5 Clone

Projects can be cloned from other projects. A project that can be cloned is called a template and it is represented with the ***is_template*** flag in the **Projects** table.

4.4 Deployment Service

This is the backend service that is responsible for deploying a project onto the cloud infrastructure.

The service is built using Flask [26], a micro web framework written in Python. It is lightweight and using a framework like Django seemed unneeded for this service. To create components on cloud infrastructure, the service uses Pulumi, an open-source software that can create, deploy, and manage infrastructure on any cloud using programming languages and tools. Since the service is written in Python, it makes use of Pulumi's Python SDK. The prototype allows the project to be deployed onto Amazon Web Services and Google Cloud Platform cloud providers.

4.4.1 Overview of Pulumi

Pulumi is a modern infrastructure as a code platform. It leverages existing programming languages like TypeScript, JavaScript, Python, Go, and their native ecosystem to interact with cloud resources through the Pulumi SDK. Fig. 8 describes the high-level architecture of Pulumi [27].

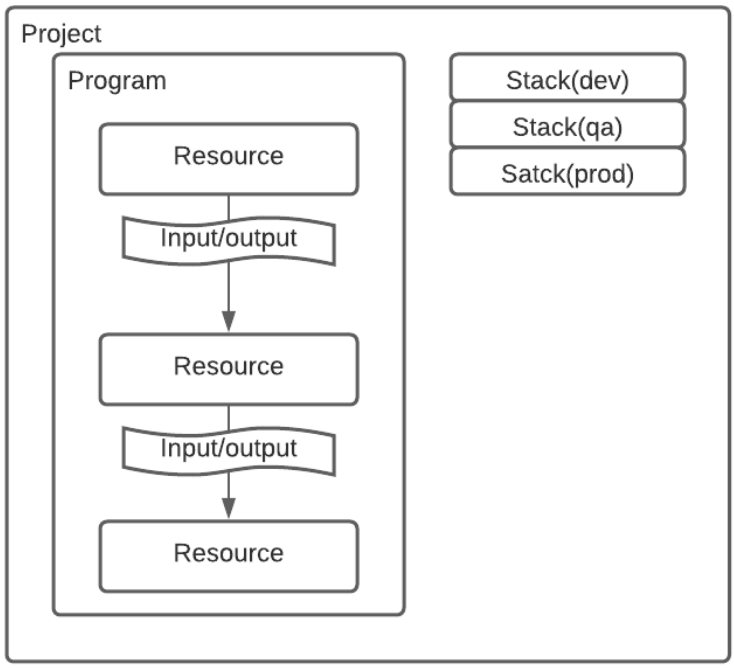

Fig. 8. Pulumi system architecture.

4.4.1.1 Pulumi Programs. These programs are written in a general-purpose language which is Python in our implementation. They describe how cloud infrastructure should be created. The program allocates resource objects whose properties correspond to the desired state of the cloud infrastructure. Pulumi programs reside in a project. Each project contains various stacks which is an instance of a Pulumi program. A stack is similar to different deployment environments that can be used when testing and rolling

out application updates. The below code shows an example of creating a security group and an EC2 instance on AWS cloud infrastructure.

```
import pulumi
import pulumi_aws as aws

group = aws.ec2.SecurityGroup('web-sg',
    description='Enable HTTP access',
    ingress=[{
        'protocol': 'tcp',
        'from_port': 80,
        'to_port': 80,
        'cidr_blocks': ['0.0.0.0/0']
    }])
server = aws.ec2.Instance('web-server',
    ami='ami-6869aa05',
    instance_type='t2.micro',
    vpc_security_group_ids=[group.name]
)
```

4.4.1.2 Pulumi Setup. The Deploy backend service talks to Pulumi through a binary installed on the server node locally and Pulumi talks to the cloud infrastructure. For the end-to-end flow to work, both Pulumi and cloud service credentials need to be set up as follows.

1) Create a user account on Pulumi
2) Install Pulumi locally on the server node
3) Log into Pulumi account

4) Setup cloud credentials through ENV variables that the local Pulumi program can access. For example, for Amazon Web Services they are AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY

For the project created from the prototype to be in sync with the Pulumi project, there is a one-to-one mapping set up through the project's ID. For simplicity, the stack name on a Pulumi project is always called "production" which roughly means production environment.

4.4.2 Why Pulumi Over Other Platforms

This section briefly talks about why Pulumi was chosen over other infrastructure as code services. It compares Pulumi against the two most popular ways used to manage infrastructure resources.

4.4.2.1 Terraform.

1) Terraform requires users to learn a new custom language, the HashiCorp Configuration Language. In contrast, Pulumi programs are written in familiar languages like Python, Javascript etc. This cuts down the learning curve steeply.
2) Terraform, by default, requires that you manage concurrency and state manually, by way of its state files.
3) Concurrency and state management are significantly easier in Pulumi. Terraform, by default, requires the user to manage it through state files. Pulumi eliminates these concerns for the user by managing them internally.
4) Pulumi supports cloud-native technologies like Kubernetes and supports advanced deployment services which cannot be expressed with Terraform.

4.4.2.2 Cloud SDKs. Cloud providers like Amazon Web Services and Google Cloud Platform offer SDKs like AWS Boto in familiar languages to create and manage cloud resources. However, managing concurrency, failures, and recovery is left to the

users to manage. Also deploying multi-cloud environments would require a considerable amount of effort and is very hard to build a robust solution.

4.5 YAML Representation

As mentioned in the frontend application section, users can create system design from a YAML file. The structure of the file is shown below.

```
project:
  name:
  description:
standard_components:
  - id:
    description:
    image:
    name:
    type:
project_components:
  - id:
    standard_component_id:
    config:
      database_selection:
      database_tag:
      instance_tag:
      input_port_number:
      output_port_number:
      load_balancer_name:
      load_balancer_selection:
      cluster_name:
```

```
      cluster_docker_image:
      cluster_minimum_size:
      cluster_maximum_size:
connections:
  - from_project_component_id:
    to_project_component_id:
```

5 COMPARATIVE ANALYSIS

From the literature survey and prototype tool developed, we now present a comparative analysis of all the features that defines a good system design. The below Table 7 indicates the features against multiple tools that were analyzed as part of this research.

1) Yes - Indicates the support of the feature
2) No - Indicates it does not support the feature
3) Partially yes - Indicates it does not completely support this feature with respect to system design
4) Future Work - This feature has been taken into consideration but yet to be implemented.

Table 7
Comparative Analysis

Features/Tools	**Microsoft Word**	**Lucidchart**	**Draw.io**	**AWS Cloud Formation**	**Prototype tool**
Based on a Standard	No	No	No	No	Yes
Sharable	Partially yes	Partially yes	Partially yes	Yes	Yes
Version control	Partially yes	Partially yes	Partially yes	Yes	Yes
Collaborative	Partially yes	Partially yes	Partially yes	Yes	Yes
Multi-level architecture	No	No	No	No	Yes
Cost evaluation	No	No	No	Yes	Yes
Performance estimation	No	No	No	Yes	Future Work
Abstract components	No	No	No	No	Yes
Automated deployment	No	No	No	Yes	Yes
Guided Process	No	No	No	No	Yes
System design as Code	No	No	No	No	Yes
Infrastructure as Code	No	No	No	Yes	Yes
Graphical representation from configuration file	No	No	No	Yes	Yes
Multi-cloud deployment	No	No	No	No	Yes
Integration with other tools	Yes	Yes	Yes	No	Future Work
Infrastructure maintenance	No	No	No	Yes	Yes

6 CONCLUSION

As part of this research, we aim to understand the shortcomings of system design. To understand the shortcomings, we first understand the various definitions of system design and how they differ from each other. The term “system design” is highly overloaded and people often exploit this to modify the representation of the system design accordingly. As a result, there is a lack of a standard. To understand the state of the art ways of creating system designs a survey was conducted on the current tools used to create system design. Analyzing the drawbacks of the tools led to the motivation of providing a standard for creating system designs along with modern-day features like enabling version control, sharing, collaboration, and automated deployment. The implemented prototype tool supports all the features that are stated as part of the solution. This enables the users to create a system design backed by a standard, enable version control, sharing, multi-cloud deployment, and also enables to create a system design from code. Overall, the prototype model aims to enable the creation of a modern system design.

7 FUTURE WORK

As part of this research, we have taken the first step towards standardizing the system design. However, defining a standard that can be universally accepted requires consideration of a wide variety of parameters, evaluating multiple use cases, higher security, increased configuration parameters, and constraints. The end-to-end application support will be highly beneficial as the application can be deployed via the tool. A set of performance test suites can be developed to enable stakeholders to make better decisions about the infrastructure requirements. As the infrastructure components increase it is important to keep up with the growing technology to meet the industry requirements. As an educational tool, additional features like enabling grading of the system design, detailed guided procedure can be beneficial. In organizations, program managers can benefit by tracking the development progress with the deadlines and providing feedback to the engineers. Improved notifications system will help all the teams to keep up with the upcoming deadlines. It is important to enabling the accessibility feature of the prototype tool. Currently, we allow for graphical and textual ways of creating system design. To enable accessibility, features like audio inputs, language translation, speech to system design creation can be implemented. Therefore, we conclude that there is always scope for improvement and development of newer features. However, this paper tries to address the core features of the system design by proposing a prototype model that initiates a way to solve the above-mentioned issues.

Literature Cited

[1] I. ISO, "Ieee, systems and software engineering–vocabulary," *IEEE computer society, Piscataway, NJ*, vol. 8, p. 9, 2010.

[2] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT Software engineering notes*, vol. 17, no. 4, pp. 40–52, 1992.

[3] J. Baragry and K. Reed, "Why we need a different view of software architecture," in *Proceedings Working IEEE/IFIP Conference on Software Architecture*, pp. 125–134, IEEE, 2001.

[4] L. Bass, P. Clements, and R. Kazman, "Architecture in practice, sei series in software engineering," 1998.

[5] P. B. Kruchten, "The 4+ 1 view model of architecture," *IEEE software*, vol. 12, no. 6, pp. 42–50, 1995.

[6] C. Hai-Shan, "Survey on the style and description of software architecture," in *8th International Conference on Computer Supported Cooperative Work in Design*, vol. 1, pp. 698–700, IEEE, 2004.

[7] N. Medvidovic and R. N. Taylor, "Software architecture: foundations, theory, and practice," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2, pp. 471–472, IEEE, 2010.

[8] P. C. Clements and L. M. Northrop, "Software architecture: An executive overview.," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1996.

[9] "Lucidchart - edutech wiki." http://edutechwiki.unige.ch/en/Lucidchart.

[10] "Lucidchart." https://www.lucidchart.com/pages/examples/uml_diagram_tool.

[11] "draw.io - crunchbase company profile & funding." https://www.crunchbase.com/organization/draw-io.

[12] "Introduction to diagrams.net and types of diagrams." https://www.diagrams.net/doc/getting-started-diagram-types.

[13] "diagrams.net documentation." https://www.diagrams.net/doc/.

[14] "diagrams.net." https://app.diagrams.net/.

[15] "History of microsoft word - wikipedia." https://en.wikipedia.org/wiki/History_of_Microsoft_Word.

[16] "Aws cloudformation." https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html.

[17] "Estimating the cost of your aws cloudformation template project." https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/tkv-cfn-editor-estimate-cost.html.

[18] "Git - wikipedia." https://en.wikipedia.org/wiki/Git.

[19] "The importance of standards - cen-cenelec." https://www.cencenelec.eu/research/tools/ImportanceENs/Pages/default.aspx#:~:text=Standards%20provide%20people%20and%20organizations,facilitating%20business%20interaction.

[20] "Standards: What are they and why are they important - standards - libguides at university of massachusetts amherst." https://guides.library.umass.edu/c.php?g=719645&p=5126968.

[21] "The advantages and disadvantages of shared documents." https://blog.mesltd.ca/the-advantages-and-disadvantages-of-shared-documents#:~:text=Document%20sharing%20saves%20time%2C%20and,development%20and%20flow%20of%20ideas.

[22] "Why use version control? — learn version control with git." https://www.git-tower.com/learn/git/ebook/en/command-line/basics/why-use-version-control/.

[23] "React – a javascript library for building user interfaces." https://reactjs.org/.

[24] "Home - django rest framework." https://www.django-rest-framework.org/.

[25] "The web framework for perfectionists with deadlines — django." https://www.djangoproject.com/.

[26] "Welcome to flask — flask documentation (1.1.x)."
https://flask.palletsprojects.com/en/1.1.x/.

[27] "Architecture & concepts — pulumi." https://www.pulumi.com/docs/intro/concepts/.

Appendix A

PROTOTYPE TOOL

The below are the screenshots of the creating a system design using the prototype tool.

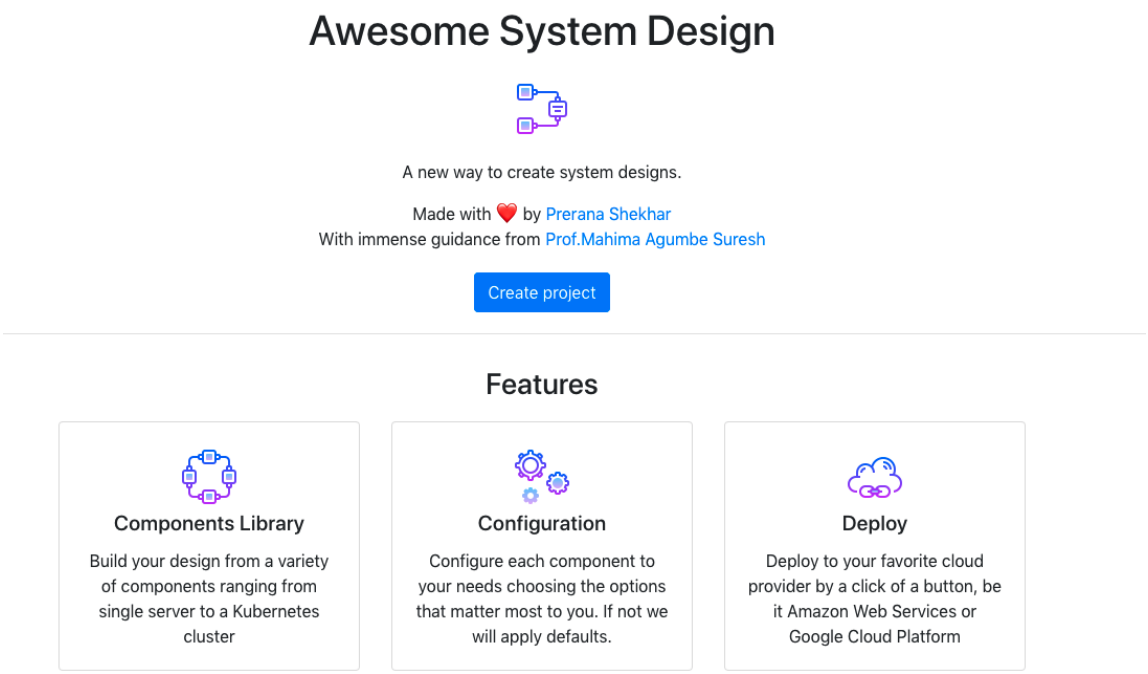

Fig. 9. Home page of prototype tool

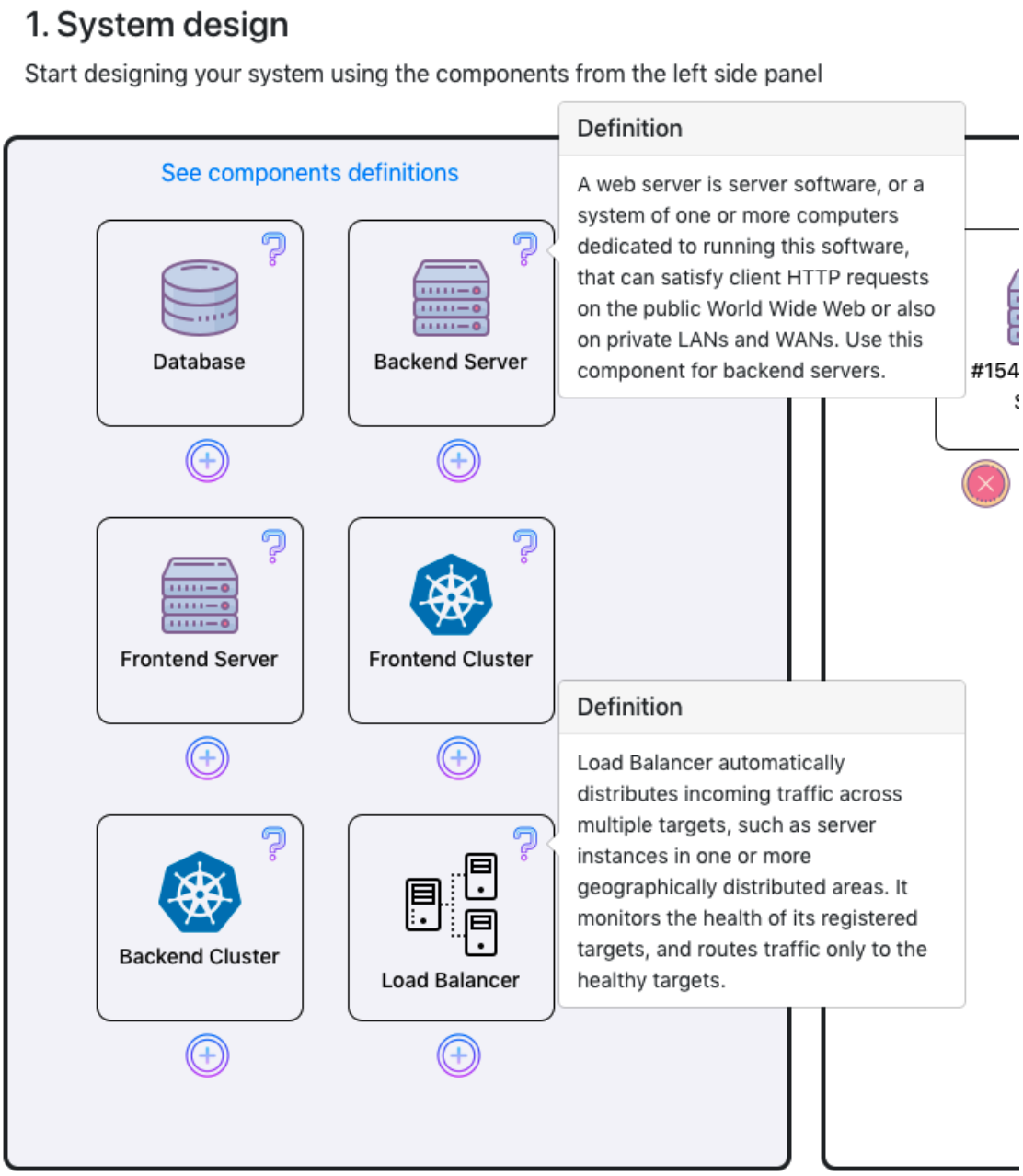

Fig. 10. Standard components of prototype tool

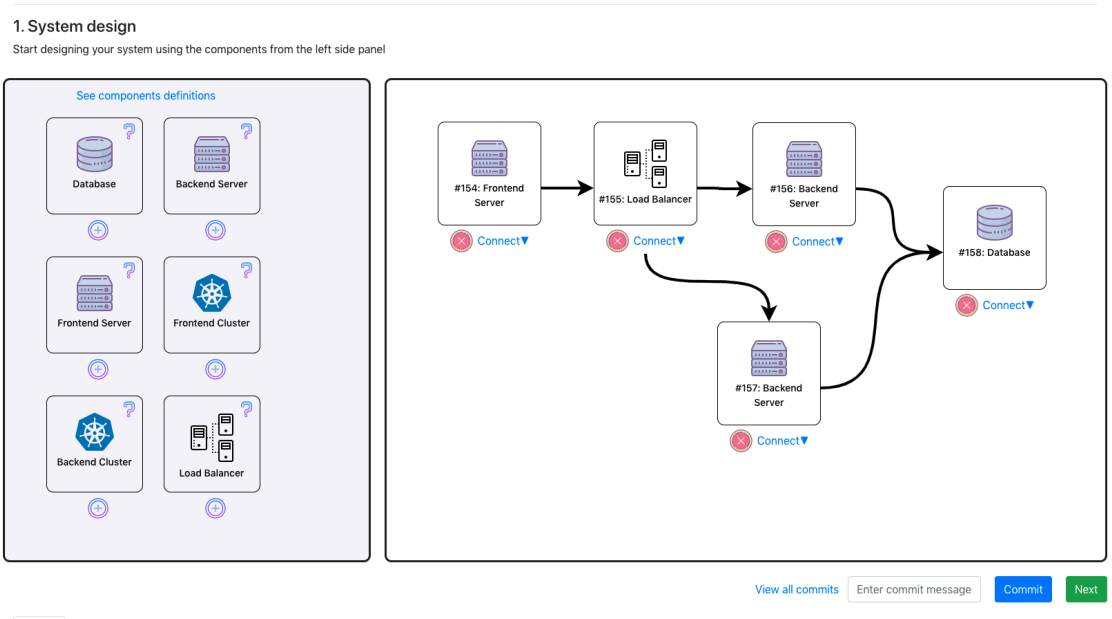

Fig. 11. System design canvas of prototype tool

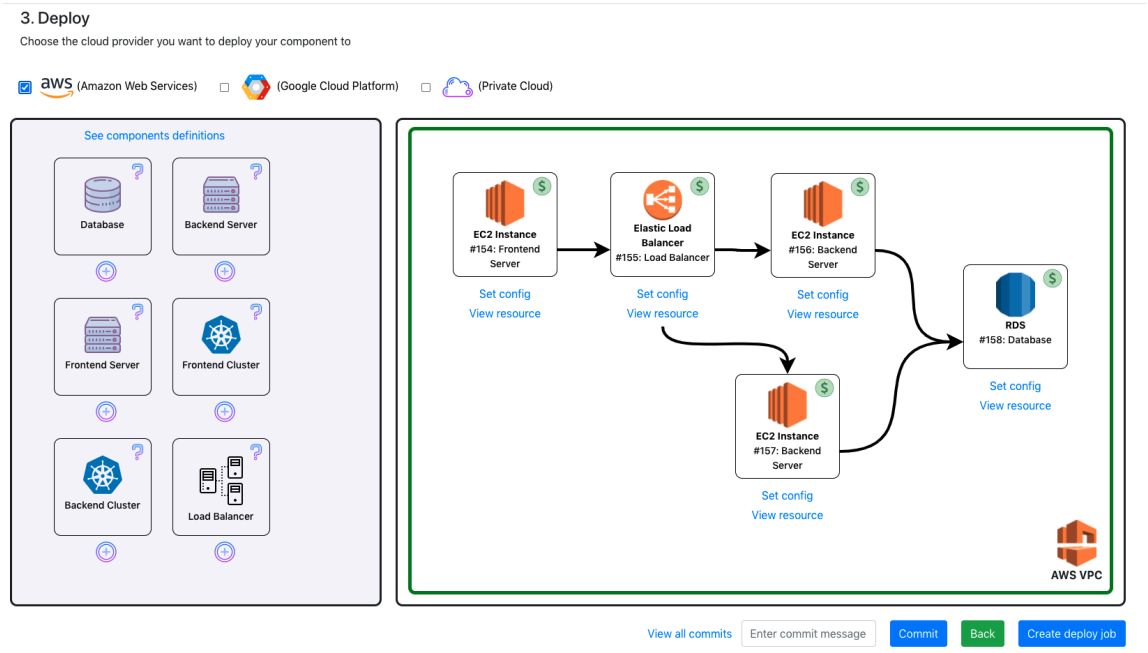

Fig. 12. AWS view of system design

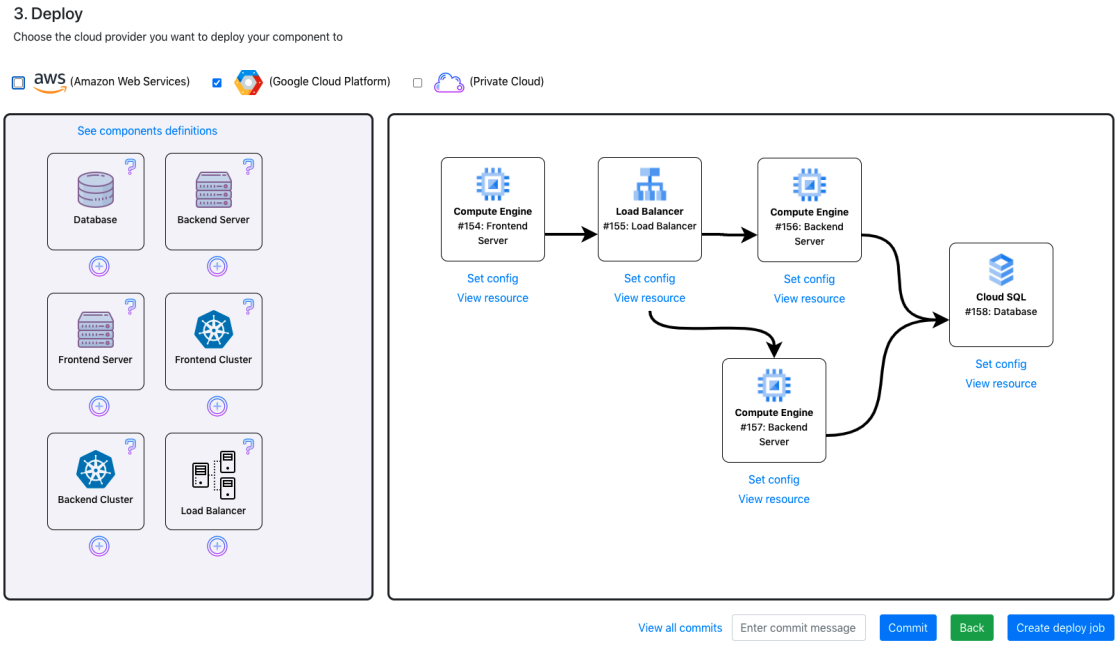

Fig. 13. GCP view of system design

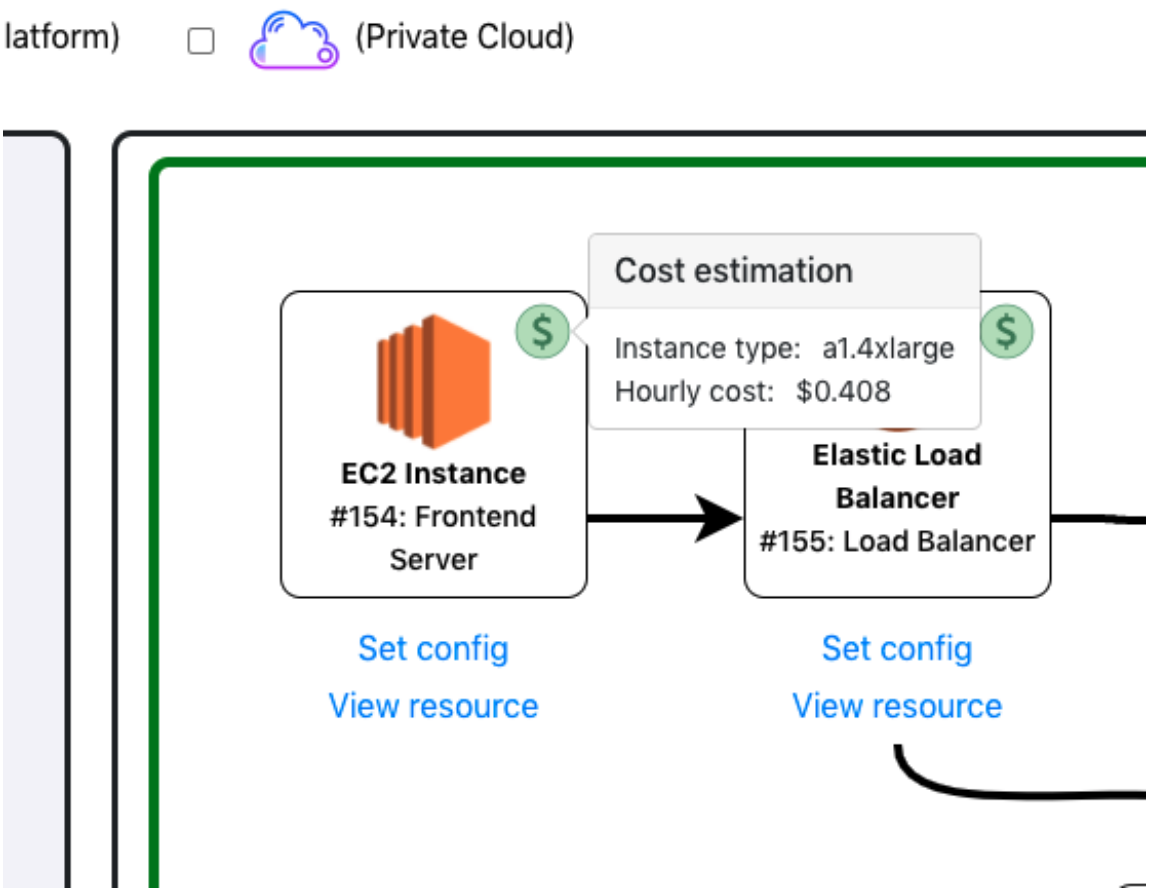

Fig. 14. Cost estimation of a component

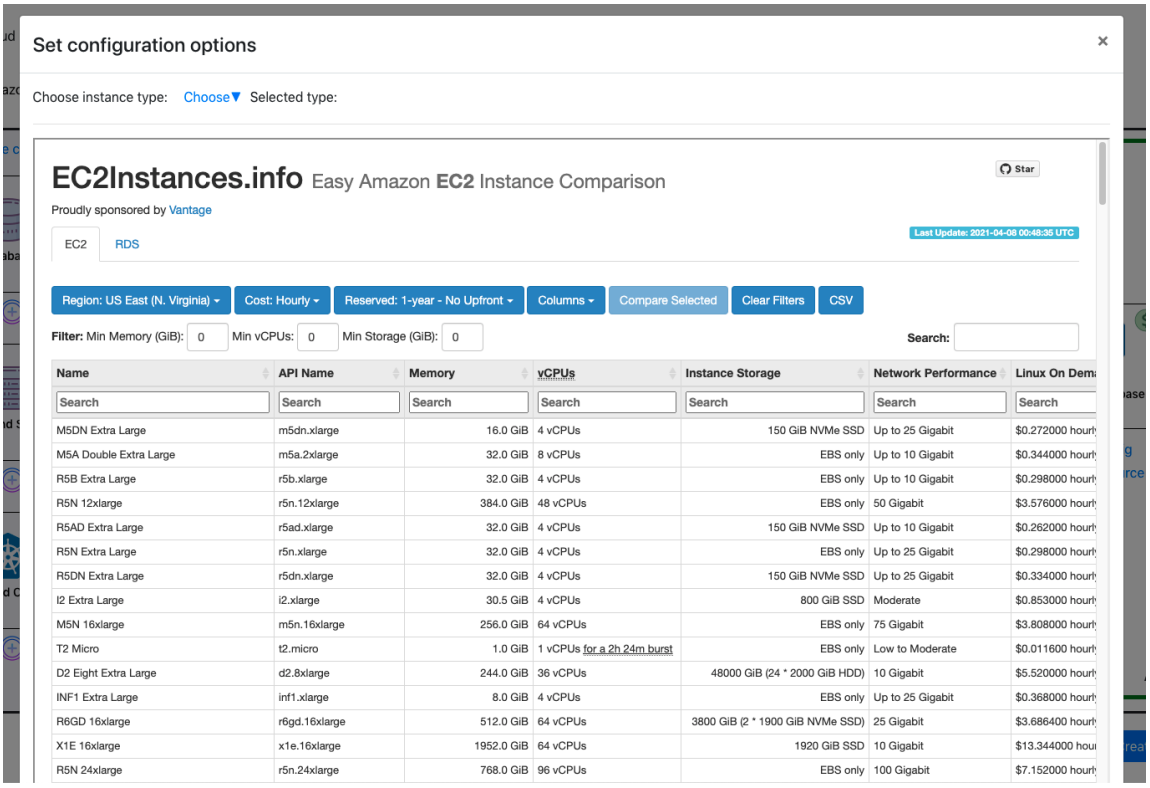

Fig. 15. AWS EC2 instance comparison

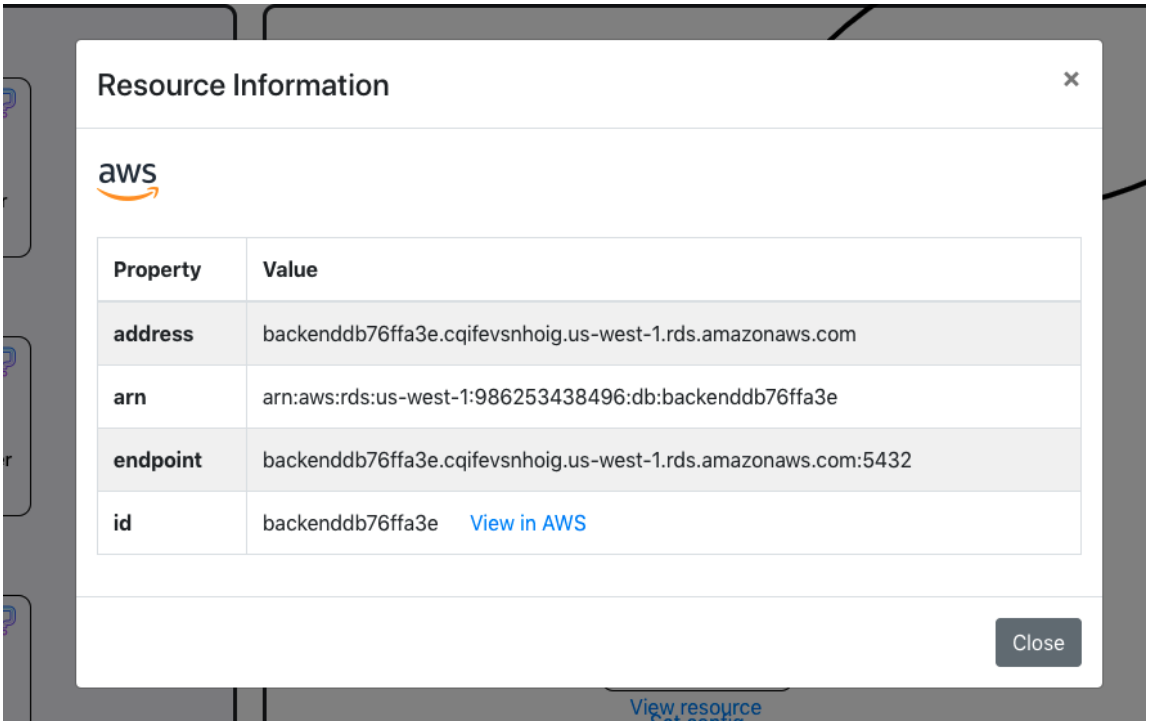

Fig. 16. AWS resource information after deployment