Universidad Michoacana de San Nicolás de Hidalgo

Facultad de Ingeniería Eléctrica
División de Estudios de Posgrado

A SOFTWARE ARCHITECTURE FOR INTELLIGENT TIME SERIES FORECASTING BASED ON CLOUD COMPUTING

TESIS

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Presenta

José Luis García Nava

Director de Tesis
Dr. Juan José Flores Romero

Morelia, Michoacán. Junio de 2018

A SOFTWARE ARCHITECTURE FOR INTELLIGENT TIME SERIES FORECASTING BASED ON CLOUD COMPUTING

Los Miembros del Jurado de Examen de Grado aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de ***José Luis García Nava.***

Dr. Félix Calderón Solorio
Presidente del Jurado [illegible]

Dr. Juan José Flores Romero
Director de Tesis [illegible]

Dr. Jaime Cerda Jacobo
Vocal [illegible]

Dr. José Antonio Carena Ibarrola
Vocal [illegible]

Dr. Carlos Alberto Lara Álvarez
Revisor Externo (CIMAT, Zac) [illegible]

Dr. Félix Calderón Solorio
Jefe de la División de Estudios de Posgrado
de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas). [illegible]

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Junio 2018

Para María del Rosario, María de la Luz, María Paz y Rosa María.

Para Juan y María Guadalupe.

Para los doctores Juan José Flores Romero, José Antonio Camarena Ibarrola, Leonardo Romero Muñoz, Jaime Cerda Jacobo y Félix Calderón Solorio.

Para todos los miembros de la comunidad de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica, Universidad Michoacana de San Nicolás de Hidalgo, con agradecimiento y respeto.

Para todos mis colegas en la transdisciplina, que no dudemos en saltar fronteras una y otra vez.

José Luis García Nava's M.Sc. program has been funded by CONACYT Scholarship No. 737505/596109.

Contents

List of Figures

Glossary

Anaconda A widely used free Python distribution with almost 200 packages for science, math, engineering, and data analysis. 109, 111, 127

Analytics-as-a-Service A data analytics software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. 1

Apache Spark An open-source powerful distributed querying and processing engine that provides the flexibility and extensibility of MapReduce but at significantly higher speeds: up to 100 times faster than Apache Hadoop when data is stored in memory and up to 10 times when accessing disk. 111, 121, 123, 128

architecture description A strictly defined work product used to express an architecture in terms of viewpoints, views, correspondences, correspondence rules and rationales. 13, 14, 21–25, 27, 28

Bokeh An interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of versatile graphics, and to extend this capability with high-performance interactivity over very large or streaming datasets. 118, 121, 123, 126–128

boto3 A Software Development Kit for Python that allows communication with the Amazon Web Services application programming interface. 120, 127

Forecasting-as-a-Service A forecasting software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. 1, 10, 32, 56, 59, 131

Infrastructure-as-Code Is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools. 9, 126

Internet of Things Is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these things to connect and exchange data, creating opportunities for more direct integration of the physical world into computer-based systems, resulting in efficiency improvements, economic benefits and reduced human intervention. 7, 33, 135

Pandas A software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. 118, 120, 123, 127

TensorFlow A software library developed by Google Brain Team within Google's Machine Learning Intelligence research organization, for the purposes of conducting machine learning and deep neural network research. 100, 111, 120, 121, 128

Acronyms

ACO Ant Colony Optimization. 35, 36

AMI Amazon Machine Image. 100, 118, 120, 121, 127

ANN Artificial Neural Network. 34

API Application Programming Interface. 5, 20, 82, 98, 114

ARIMA Auto Regressive Integrated Moving Average. 34

ARIMAX Auto Regressive Integrated Moving Average with Exogenous Input. 34

AWS Amazon Web Services. x, 8, 9, 12, 95–100, 102–121, 123, 124, 126, 128, 132–134

AZ Availability Zone. 96, 109, 111, 114, 115

BPMN Business Process Model and Notation. 66

CACO Chaos/Cloud Ant Colony Optimization. 36

CGA Chaos/Cloud Genetic Algorithm. 36

CIA Chaos/Cloud Immune Systems. 36

CPSO Chaos/Cloud Particle Swarm Optimization. 36

CPU Central Processing Unit. 58, 59, 65, 78, 121, 123, 127

CRUD Create, Read, Update, Delete. 126

CSA Chaos/Cloud Simulated Annealing. 36

CSV Comma-Separated Values. 75, 103, 127

CTA Chaos/Cloud Tabu Search. 36

DE Differential Evolution. 37, 38

DNS Domain Name System. 98, 114

EBS Elastic Block Storage. 97, 104, 107, 109, 121, 127

EC2 Elastic Cloud Computing. x, 96–100, 102, 103, 107, 109, 111, 114, 115, 120–124, 126, 127

ELB Elastic Load Balancing. 96

ELFSS Electric Load Forecasting Support System. 35, 36, 38

EMR Elastic MapReduce. 99, 118, 121, 123

FTPS File Transfer Protocol Secure. 59

GA Genetic Algorithm. 35, 36

HTTP Hypertext Transfer Protocol. 59, 82, 99, 114, 115, 124

HTTPS Hypertext Transfer Protocol Secure. 59, 124

IA Immune Systems. 35, 36

IAM Identity and Access Management. 98, 102, 114, 127

IP Internet Protocol. 93, 96, 103, 124, 126, 127

IT Information Technology. 2, 51, 53, 54, 65, 133

ITSFCC Intelligent Time Series Forecasting based on Cloud Computing. v, vi, viii–x, 4, 6, 10–13, 25–30, 36–41, 44–50, 56–75, 78, 82, 86, 89, 93, 95–119, 121, 123, 128, 131–136

JSON JavaScrip Object Notation. 75, 103

KBES Knowledge-Based Expert System. 34

KMS Key Management Service. 102

KNN K-Nearest Neighbors. 121

NAT Network Address Translation. 126

NIST National Institute of Standards and Technology. 50, 53

NN Nearest Neighbors. 37

NNDE Nearest Neighbors - Differential Evolution. 36–38, 40, 41, 47, 134

NoSQL Not only SQL. 75, 97, 103

PSO Particle Swarm Optimization. 35, 36

RDS Relational Database Service. 97, 102, 104, 107, 109, 120, 124, 126

REST Representational State Transfer. 55, 114, 124

RFID Radio-Frequency Identification. 47

S3 Simple Storage Service. 97–99, 102–104, 107, 109, 118, 120, 123, 127

SA Simulated Annealing. 35, 36

SARIMA Seasonal Auto Regressive Integrated Moving Average. 34

SDK Software Development Kit. 120

SMS Short Message Service. 127

SNS Simple Notification Service. 99, 127

SOA Service-Oriented Architecture. 54, 55

SOAP Simple Object Access Protocol. 55

SQL Structured Query Language. 75, 99, 121, 123

SQS Simple Queue Service. 99, 104, 107, 109, 111, 115

SRV Support Vector Regression. 34–36

SSH Secure Shell. 93, 123, 124

SSL Secure Sockets Layer. 59

TA Tabu Search. 35, 36

TLS Transport Layer Security. 59

TOGAF The Open Group Architecture Framework. 134

URI Uniform Resource Identifier. 55, 60, 114

URL Uniform Resource Locator. 120

VPC Virtual Private Cloud. 98, 102, 103, 124, 126

XML Extensible Markup Language. 55, 75, 103

YAML YAML Ain't Markup Language. 89, 126

Abstract

Cloud computing enables research institutions to transfer specialized knowledge to external users, as well as to collaboratively work among interdisciplinary, multi-location teams. However, cloud computing is not widely used in mexican research institutions yet. In the case of the research Work Group where this thesis was conceived, a methodology is required to properly deploy its software research products to the cloud.

This thesis proposes the design of an architecture for a software system able to integrate the Work Group's research products as a cloud-native application. In accordance with the research lines driven by the Work Group, this software system is intended to apply Artificial Intelligence and Machine Learning-based forecasting methods to produce high-quality predictions for time series describing multiple-domain variables. In order to formally express this architecture, a characterization based on the standard for architecture description was designed. This characterization defines an architecture model that is oriented to a process view, expressed via block diagrams, and based on software patterns.

On this ground, a cloud-native, pattern-based, provider-independent architecture was built. This architecture comprises 15 application components designed to provide cloud functionality to the Work Group's research products. In addition, a reference architecture for deploying the system on the Amazon Web Services cloud was designed.

Keywords: machine learning, software patterns, Forecasting-as-a-Service, Amazon Web Services.

Resumen

La computación en la nube permite a las instituciones de investigación transferir conocimiento altamente especializado hacia usuarios externos, así como colaborar en equipos interdisciplinarios, basados incluso en diferentes ubicaciones. Sin embargo, el uso de la computación en la nube todavía no está generalizado en las instituciones de investigación mexicanas. En el caso del Grupo de Trabajo en el que esta tesis fue concebida, se carece de una metodología para trasladar el software resultado de sus investigaciones a la nube.

Esta tesis propone el diseño de una arquitectura para un sistema de software que integre los productos de investigación del Grupo de Trabajo en la forma de una aplicación nativa para la nube. En concordancia con las líneas de investigación del Grupo de Trabajo, este sistema tiene por objetivo utilizar métodos de pronóstico basados en inteligencia artificial y aprendizaje automático para producir predicciones de alta calidad para series de tiempo que describan variables de múltiples áreas de conocimiento. Para formalizar esta arquitectura se diseñó una caracterización basada en el estándar para descripciones de arquitectura de sistemas. Esta caracterización define un modelo de arquitectura que está orientado hacia una vista de procesos, expresado mediante diagramas de bloques y basado en patrones de software.

En este contexto se construyó una arquitectura nativa para la nube, basada en patrones e independiente de proveedor. Esta arquitectura comprende 15 componentes de aplicación diseñados para proveer funcionalidad en la nube a los productos de investigación del Grupo de Trabajo. Adicionalmente se diseñó una arquitectura de referencia para implantar el sistema en la nube de Amazon Web Services.

Keywords: aprendizaje automático, patrones de software, Pronóstico-como-Servicio, Amazon Web Services.

Chapter 1

Introduction

This work is part of the activities of the Data Science Research Work Group (The Work Group) at the Graduate School of Electrical Engineering (DEP-FIE), Universidad Michoacana de San Nicolás de Hidalgo (UMICH). For more than 15 years, researchers and graduate students in the Work Group have produced specialized knowledge centered on topics like Forecasting, Time Series Analysis, Artificial Intelligence, Machine Learning, and Soft Computing. This knowledge has impacted not only generations of Computer Science and Electrical Engineering students, but also non-academic entities that turned around to UMICH in search for a solid partner in advanced innovation projects.

In recent years, the Work Group started collaboration as a forecasting provider for different entities, most of them part of the mexican electric power industry. By developing and implementing Artificial Intelligence and Machine Learning-based methods of forecasting, the Work Group attempts to contribute in closing the wide gap that separates the effective tools modern data science offers from the methods currently used by mexican productive enterprises. In this context a fundamental question arises: how to efficiently transfer the high-level knowledge generated by research institutions to the productive entities that demand it? Answering this question is an arduous matter, specially in the mexican industry and services landscape, characterized by a weak culture of innovation that leads to reduced investments in research and development.

A forecasting and data analytics service offered via the Internet to multiple customers was proposed by Juan J. Flores, head researcher of the Work Group, as a suitable solution. This Analytics-as-a-Service or Forecasting-as-a-Service (FaaS) tool would enable tenants in different organizations to benefit from the specialized knowledge generated in the Work Group at a fraction of the cost usually paid for consulting services of this level. In addition, customers would not be required to have any special expertise or skill to produce high-quality forecasts or analysis, as long as Artificial Intelligence and Machine Learning-based techniques would mask the decision making process behind a clear and simple user interface.

Soon after the solution analysis began a parallel problem came up, represented by the fact that most of the Work Group's research intended to support the FaaS operation was not cloud-native, meaning it was not developed using the specific resources of the cloud computing paradigm. Although this condition does not affect the applicability or the effectiveness of algorithms or programs, it does complicate their deployment on the cloud, as long as the elements that constitute infrastructures (storage, processing, networking) and platforms (operating system, programming languages, middleware) located in a desktop environment are very different from those found in the cloud landscape. This antecedent can be explained by the fact that cloud computing is a resource just recently available to Information Technology (IT) teams, therefore technology-related communities, researchers included, are still accustomed to develop their projects as *on-premises* solutions, deployed in local infrastructures ranging from personal computers to high-performance clusters. However, the ability of cloud computing to host environments that boost collaboration, as well as its potential to stimulate a transversal, interdisciplinary way of conceptualizing software solutions are good reasons to expect a progressively higher utilization of this paradigm in research projects. As a consequence, the first driving question of this thesis, regarding knowledge transfer, found an interesting complement in a second one: how to transform research activities inside the Work Group in order to adopt cloud computing not only as an efficient way to deliver its academic products, but also as a collaboration platform able to accelerate its workflows and enhance its performance? This time, the answer pointed out to a potential expansion of the Work Group's research interests in the direction of topics like service-oriented and cloud computing, service science, distributed applications, web services, etc.

In this context, the development of a software system that takes advantage of the cloud computing paradigm to enable non-specialized users to produce high-quality forecasts, based on the research produced by the Work Group, emerged as an appropriate seed work for addressing both sides of the problem: how to leverage cloud computing as the basis of a knowledge transfer platform, as well as the foundation for collaborative research. Although the development of such a software system represents a very extensive task, and it would have placed an unattainable goal for this work, the design of its architecture is a suitable assignment, able to render high-end results within the temporal scope of a graduate thesis, and to provide a solid basis for a further, expanded research project.

1.1 Problem Definition

The main problem this thesis addresses is the lack of a methodology to integrate the models, algorithms, and programs developed by the Work Group as elements of a cloud-native application that allows to transfer knowledge outward as well as to share research procedures and results inward. This problem can be translated into action as the design of a cloud-native application architecture because of the following reasons:

- Designing the architecture encompasses the explicit integration of models, algorithms, and programs as parts of specific application components. In this context, the architecture acts as a master blueprint where the aforementioned Work Group's research products can be precisely located, along with the communication exchanges they participate in.
- Designing the application architecture is equivalent to start building the application itself, at an abstract level. This is particularly important for the Work Group's research products because most of them exist only as isolated experiments and not as elements of a unified application.
- Once the architecture is defined and essential decisions concerning the application design have been made, it can provide definite templates and procedures for further modifications, expansions, or enhancements to the initial development.
- Conway's Law states that any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communications structure [Conway, 1968]. However, organizations seeking to move to cloud-native architectures have often employed the Inverse Conway Maneuver: rather than building an architecture that matches their organization chart, they determine the

architecture they want and restructure their organization to match that architecture [Stine, 2015]. In the context of the Work Group, this means that the design of this architecture can be regarded not only as a way to consolidate isolated research products, but also as a directive for a further alignment of human resources towards a unified, collaborative research strategy.

In summary, the design of a cloud-native application architecture as a way for migrating isolated research products like models, algorithms, or programs to a cloud-computing environment can be regarded as a strategy that allows academic institutions to enhance its production through collaboration as well as to stand out as suitable research and development partners for other organizations in the same knowledge ecosystem. The significance of solving the problem defined in these lines is based on the fact that the overall conditions described are not exclusive of the Work Group, but widely spread across mexican research entities.

1.2 Context

This section briefly presents cloud-native application architectures as the context for Intelligent Time Series Forecasting based on Cloud Computing (ITSFCC) architecture design and proposes the integration of hybrid information resources as an adequate approach to the problem that drives this thesis.

1.2.1 Cloud-Native Application Architectures

Designing a software system for a cloud environment is not equivalent to just bundle an on-premises application as a monolithic virtual machine or container [Leymann et al., 2016]. It demands a completely different approach able to take into account not only the motivations for moving operations to the cloud, but also the way this migration impacts traditional development practices. For instance, an efficient cloud computing service makes scalability and parallelization effortless, then computing problems that place a high demand for performance can be addressed by launching a high number of cloud-based parallel resources, which are available at a low cost from a practically limitless pool, instead of by meticulously optimizing the static, serial computing-based resources already owned. This is not a minor change.

It impacts budget assignment as long as the investment in hardware and facilities (capital expenditure, or CapEx) decreases while the investment in cloud and consulting services (operating expenses, or OpEx) increases. It also affects research and development workflows because human resources previously assigned to hardware and software configuration can now be shifted to domain-specific tasks closer to the entity's business logic. As a result, services can be rendered at a higher speed and scale. Stine examines the following characteristics of cloud-native application architectures and how they can provide speed, safety, scale, and mobility to business via software [Stine, 2015]:

- *Twelve-factor app* model. A collection of patterns[1] intended to optimize application design for speed, safety, and scale by emphasizing declarative configuration, stateless/share-nothing processes that horizontally scale, and an overall loose coupling to the deployment environment [Wiggins, 2014].

- Microservices-based. Microservices are an architecture pattern that helps enterprises to align its units of deployment with business capabilities, allowing each capability to move independently and autonomously, with enhanced speed and safety [Lewis and Fowler, 2014].

- Self-service agile infrastructure. Cloud platforms that enable development teams to operate at an application and service abstraction level, providing infrastructure-level speed, safety and scale.

- API-based collaboration. An architecture pattern that defines service-to-service interaction as automatically verifiable contracts, enabling speed and safety through simplified integration work.

- Antifragility. As stress on the system is increased via speed (the ability to innovate, experiment, and deliver value more quickly than competitors) and scale (the ability to elastically respond to changes in demand), the system improves its ability to respond, increasing safety (the ability to move rapidly but also maintain stability, availability, and durability) [Taleb, 2012], [Tseitlin, 2013].

[1]A body of work known collectively as architectural patterns or styles has catalogued a number of successfully used structures that facilitate certain kinds of component communication. These patterns are essentially reusable architectural blueprints that describe the structure and interaction between collections of participating components[Gorton, 2006]

In terms of defining the landscape of general software architecture it is difficult to go past a few key books from members of the Software Engineering Institute written in the early 2000s [Gorton, 2006]. The vast research on general software architecture produced during these almost two decades has recently expanded towards cloud-native application architectures due to the rise of cloud computing and the successful business model it has driven in companies like Facebook, Uber, Netflix, or Airbnb. However, the aforementioned research does not offer close references to the very specific problem addressed in this thesis, as long as no other research project working on a cloud-native architecture for the application of Artificial Intelligence and Machine Learning-based methods to time series forecasting was found when searching for architecture or implementation references. Besides the specificity of the problem, a couple of circumstances contribute to this lack of references:

- From the extensive revision on the state of the art presented in [Hong, 2013] it can be noticed that most of the results provided by researchers in forecasting, time series forecasting, and intelligent time series forecasting are focused on developing models and algorithms, but not on the resources that internally support such developments, like software architectures, software patterns, or data engineering.

- On the other hand, cloud-native architectures are produced, to a great extent, as the work of specific cloud computing providers that tend to prioritize research on solutions to problems that represent higher business opportunities for them, leaving other topics unattended.

These two circumstances unveil the fact that the intersection of cloud-computing with intelligent-based methods for time series forecasting produces an emergent, interdisciplinary research field which demands to be addressed by deploying unprecedented strategies apt to hybridize different types of resources. On this ground, the design of ITSFCC architecture is not based on close architecture research or reference architectures, but on the integration of heterogeneous resources coming from multiple environments. These resources will be enumerated and briefly described in the next chapter as an attempt to configure a state of the art for the problem addressed in this thesis.

1.3 Objectives

This section presents the general objective that drives this work, as well as the specific objectives derived from it.

1.3.1 General Objective

To design a cloud-native, pattern-based, abstract architecture for Intelligent Time Series Forecasting based on Cloud Computing (ITSFCC), a software system intended to produce high-quality forecasts for time series describing multiple variables, with a special focus on the electric power industry, as well as to validate the functionality of this architecture by mapping its components to a specific public cloud computing offering.

1.3.2 Specific Objectives

The specific objectives derived from the aforementioned general objective are:

- To define or to select a model suitable to express the ITSFCC architecture as the final product of this thesis, as well as the basis for future research work. In order to achieve this objective, basic research must be conducted on the fields of software architecture and software patterns.
- To analyse ITSFCC's operation context in relation to the Work Group's essential research domain, which is the application of Artificial Intelligence and Machine Learning-based techniques to time series analysis and forecast model production, as well as to define the requirements this context places to the ITSFCC architecture.
- To analyse ITSFCC's operation context in terms of the cloud computing paradigm application, as well as to define the requirements this context places to the ITSFCC architecture.
- To elaborate an abstract, provider-independent, pattern-based architecture for ITSFCC that addresses all of the requirements issued by the analysed context of ITSFCC operation.
- To map the abstract architecture designed for ITSFCC to a specific cloud offering as a way to validate the adequacy of the architecture

model to concrete cloud resources and implementation procedures. In order to achieve this objective, a provider-specific, general architecture model will have to be produced, as well as tested for the deployment of its essential components.

1.4 Thesis Overview

This remainder of this thesis is organized as follows:

Chapter 2 presents the information resources the design of ITSFCC architecture is based on, as an attempt to configure a state of the art for this thesis. The concept of software architecture is introduced to provide a methodological basis for subsequent chapters. The standard definition of architecture is presented, as well as a series of characteristics that expand this definition towards a software-related environment. Software quality attributes are discussed as they are considered fundamental elements for a software architecture characterization. The International Standard ISO/IEC/IEEE 42010 for systems and software architecture is summarized, and then used as a basis for the definition of an architecture characterization for ITSFCC.

Chapter 3 introduces the concept of intelligent time series forecasting and examines the architectural elements a software system requires to address it. The ITSFCC environment is described in terms of the high complexity that characterizes forecasting in the energy domain, and the Artificial Intelligence and Machine Learning-based methods developed to deal with such complexity. Concerns placed to ITSFCC under this environment are identified and expressed as broad features the system must include. These features are mapped to ITSFCC application components, and then to precise software quality attributes. Finally, software patterns suggested to deal with the identified concerns are described.

Chapter 4 presents the concept of cloud computing and examines the basic architectural elements an intelligent time series forecasting application must implement to leverage cloud computing potentials. ITSFCC environment is described in terms of the standard definition of cloud computing and the technological and organizational prerequisites for its adequate implementation. Concerns placed to ITSFCC under this environment are identified and expressed as broad features the system must include. These features are mapped to ITSFCC application components, and then to precise software quality attributes. Finally, software patterns suggested to deal with the identified concerns are described.

Chapter 5 presents the detailed, architecture characterization of ITSFCC, in compliance with the specifications covered in chapter 2. A set of text-based documents containing the architecture identification and overview, the identification of stakeholders, and the identification of concerns is presented. An architecture model that conforms to a process-based architecture view and is governed by a block diagram model kind is also presented. The overall architecture of ITSFCC, as well as each one of its application components are depicted in a comprehensive set of diagrams, which are described in detail by explanatory texts.

Chapter 6 departs from the abstract architecture characterization of ITS-FCC presented in chapter 5, and maps it to the services offered by a specific public cloud provider. Amazon Web Services (AWS) was selected as the concrete cloud offering to land ITSFCC's abstract architecture on because of the extension and maturity of its services, as well as for the facilities it offers to deploy cloud resources on a trial basis. An overview of AWS and the services required to implement ITSFCC components in the AWS cloud are presented. A set of guides for implementing ITSFCC components in AWS, consisting on detailed, annotated architecture blueprints is also presented. Finally, a set of experiments deployed on AWS in order to validate the adequacy of specific services to the requirements placed by ITSFCC components is summarized.

Chapter 7 presents the general and specific conclusions of this work, as well as the directives for future work.

Chapter 2

Software Architecture

This chapter introduces the concept of *software architecture* which provides a foundation for the content of subsequent chapters. Section 2.1 presents a state of the art for the design of a cloud-native architecture for intelligent time series forecasting. Section 2.2 departs from the standard definition of *architecture* and then describes a set of characteristics that expand the definition towards a software-related environment. Software quality attributes are discussed in Section 2.3 as they are considered fundamental elements for a software architecture characterization. Section 2.4 summarizes the International Standard ISO/IEC/IEEE 42010 for systems and software architecture description. Finally Section 2.5 proposes an architecture characterization for ITSFCC that will provide a methodological guide for chapters 3 and 4.

2.1 State of the Art

The problem addressed in this thesis can be considered as highly specific, as long as no other research project working on a cloud-native architecture for intelligent time series forecasting was found in the literature. This situation makes difficult to prepare a classic state of the art review because no reference works are available for a comparative basis. As a result, an attempt to configure a state of the art is made via the analysis of different categories of information resources related to the design of ITSFCC architecture.

2.1.1 Information Resources for ITSFCC Architecture Design

The information resources that provided a research background for ITSFCC architecture design are focused on the following topics:

Cloud Computing in relation to Service-oriented Computing

The compilations in [Aiello et al., 2016] and [Dustdar et al., 2015] address service-oriented computing as an important paradigm for the development of distributed software applications, with a special emphasis in the use of services in cloud infrastructures. On this ground, topics like service policies and performance, service adaptation, service level agreements, job placement, service compositionality, fault tolerance, and the Internet of Things are discussed in the collection of papers included in these works.

Cloud Computing in relation to Service Science

The work in [Castro-Leon and Harmon, 2016] provides an extensive background for understanding service science history and foundations, as well as the implications cloud computing represents for service transformation nowadays. The compilations in [Helfert et al., 2017] and [Helfert et al., 2016] contribute to the understanding of relevant trends of current research on cloud computing and service science, like cloud interoperability and migration, cloud-native applications, microservices, and containers, auditing and service level agreement management, detecting anomalies in cloud platforms, deployment and adaptation of cloud data centers, public high-performance clusters, and private cloud computing.

Cloud Computing in relation to Big Data Analytics and Applications

The fundamental technologies that conform the current Big Data landscape, such as distributed computing, data serialization, NoSQL and columnar storage, messaging systems, and distributed query engines are summarized in [Guller, 2015]. A generic Big Data application architecture is presented in [Sawant and Shah, 2014], along with useful patterns for data ingestion, data streaming, storage, access, data discovery, visualization, and deployment in a Big Data environment. Patterns that allow fine-tunning for non-functional requirements like security, performance, scalability, and availability for Big Data applications are also discused in this work. A complete revision of Apache Spark functionalities for interactive data analysis, SQL-like querying, data streaming, machine learning, and graph processing is provided in [Kane, 2017], [Drabas and Lee, 2017], and [Nandi, 2015].

Cloud Computing in relation to Software Patterns

The work in [Fehling et al., 2014] introduces a precise and complete software pattern format and uses it to present specific patterns for cloud computing fundamentals, cloud offerings, cloud application architectures, cloud application management, and composite cloud applications. By using the same pattern format to describe both the generic properties of cloud offerings and the best practices on how to deal with these properties this work promotes research products that are programming language-neutral and provider-independent. The extensive work in [Raj et al., 2017] reviews widely used patterns for software architecture, design, deployment, and integration, such as Client/Server, Multi-tier, Object-oriented, Domain-driven, Service-oriented, Event-driven, Microservices, Containerized Applications, and Big Data Architectures. By enumerating and expressing a huge collection of prominent and dominant software patterns this work points out the heterogeneity and complexity of contemporary software and data engineering.

Time Series Forecasting

The work in [Dannecker, 2015] analyzes time series forecasting in the energy domain and proposes a forecasting process that allows an application-aware and efficient calculation of accurate predictions. By introducing the concept of a context-aware forecasting model repository, this work enhances the forecast model selection and the parameter estimation stages. It also presents a framework that allows an efficient integration of external information into forecast models. The work in [Hong, 2013] analyzes traditional and Artificial Intelligence-based approaches for energy demand forecasting, and proposes a hybrid approach based on the combination of support vector regression, fuzzy logic, and evolutionary computation.

Cloud Computing in relation to the Amazon Web Services Cloud

An introduction to the best practices on how to move an enterprise-class application from a fixed physical environment to a virtualized cloud environment is given in [Varia, 2010]. However, due to the extension of the AWS ecosystem, it is advisable to consult works that address its specific functionalities in a deeper level. The most widely used services of AWS are enumerated and moderately described in [Wadia, 2016] and in [Vyas, 2015]. These works offer basic to intermediate level contents related to security, access management, virtual instances deployment, storage, networking, monitoring, databases, and application management. All of the aforementioned service categories are reviewed, with a special focus on how they relate to software quality attributes like security, scalability, availability, and performance in [Sarkar and Shah, 2015]. A description of AWS' functionalities focused on Infrastructure-as-Code is provided in [Chan and Udell, 2017]. Finally, extensive reviews can be consulted for selected services like networking [Das and Modi, 2017], or continuous integration and delivery [Kantsev, 2017].

Architectures for Time Series Forecasting

Although no other research project working on a cloud-native architecture for intelligent time series forecasting was found, a couple of close projects can be referenced. The work in [Dannecker, 2015] proposes an architecture for a common energy data management system. It includes a forecasting component that tightly integrates the time series storage with the forecasting algorithms to directly access and process time series data. The architecture of this forecasting component is extensively described and includes advanced elements such as in-memory data storage and a publication-subscription messaging system. The forecasting process in this work is based on multi-equation ARIMA models. The work in [Afanasieva et al., 2018] proposes a web application with open access aimed on forecasting of time series stored in databases. It presents a system architecture which handles forecasting model selection and an application programming interface as web services. The forecasting process in this work is based on a hybrid approach that combines ARIMA models and fuzzy techniques.

2.2 Architecture Basics

Every system, software-related or not, exhibits an architecture. Therefore a software architecture is a particular case of system architecture or, as it is commonly referred, architecture. This section presents the standard definition of architecture and expands it towards a software-related context in order to develop an inital approach to the software architecture concept. Accordingly, two categories are described: the key aspects included in the standard definition and the characteristics that result from relations the architecture establishes with the key aspects it comprises.

2.2.1 Definition of Architecture

Architecture is defined in [ISO/IEC/IEEE, 2011] as *the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.* [Pillai, 2017] enumerates the key aspects rooted to this definition and maps them to a software-related context as follows:

1. System: A system is a collection of components organized in specific ways to achieve a specific functionality. A software system is a collection of such software components. A system can often be subgrouped into subsystems.

2. Structure: A structure is a set of elements that are grouped or organized together according to a guiding rule or principle. The elements can be software or hardware systems. A software architecture can exhibit various levels of structures depending on the observer's context.

3. Environment: The context or circumstances in which a software system is built, which has a direct influence on its architecture. Such contexts can be technical, business, professional, operational, and so on.

4. Stakeholder: Anyone, a person or groups of persons, who has an interest or concern in the system and its success. Examples of stakeholders are the architect, development team, customer, project manager, marketing team, and others.

Figure 2.1 (extracted from [ISO/IEC/IEEE, 2011]) shows a class diagram with the context of an architecture description. Although the concept of architecture description has not been covered yet, the rest of the diagram clarifies the relations that emerge between a system architecture and the core elements in its formal definition. Conventions used are defined in [ISO/IEC, 2005] so the reader is advised to review this standard for a fully detailed description. The class diagram depicts the following relations:

- A system may exhibit more than one architecture.
- An architecture may be expressed by one or more architecture descriptions, or none at all.
- A stakeholder may have interests in one or more systems.
- One or more stakeholders may have interests in a system.
- Zero or more systems may be situated in an environment, but a system is situated in only one environment.

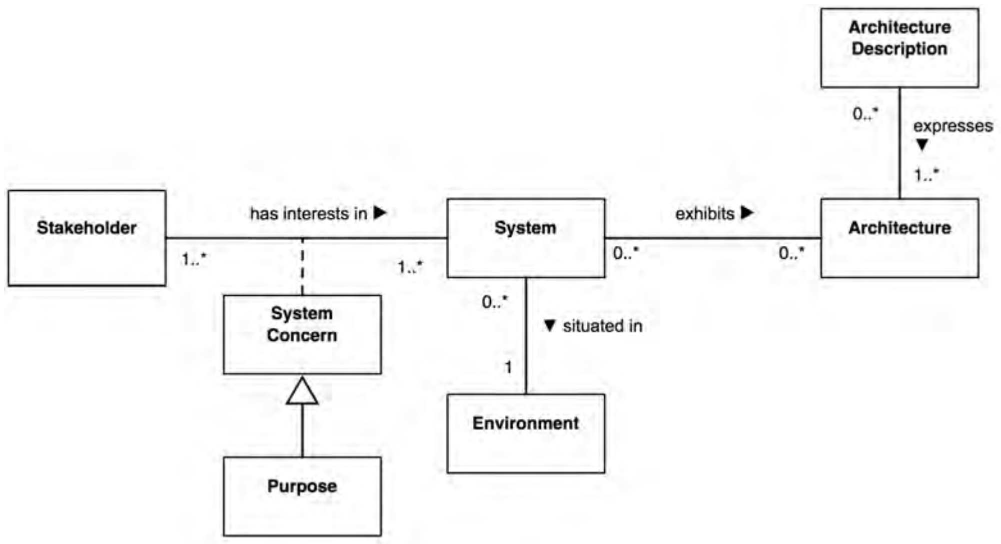

Figure 2.1: Context of architecture description [ISO/IEC/IEEE, 2011]

2.2.2 Characteristics of Software Architecture

Mapping the elements included in the definition of system architecture to their corresponding aspects in a software system is an informal but convenient way to build a first approach to the concept of software architecture. [Pillai, 2017] and [Gorton, 2006] extend this effort by enumerating a set of characteristics found in every software architecture. It is important to notice that each characteristic is based on a distinct relation between the software architecture itself and at least one of the key aspects the architecture comprises:

1. An architecture defines a structure. As a common practice, the system architecture representation conforms to a structural component or class diagram able to represent both the involved subsystems and the relationships established between them.

2. An architecture picks a core of set elements. A well-defined architecture captures only the core set of structural elements required to build the core functionality of the system and avoids documenting everything about every component of the system.

3. An architecture captures early design decisions. As long as it has to be focused on the elements required for offering the core functionality, an architecture summarizes early design decisions about the system and keeps them visible for further development processes.

4. An architecture manages stakeholder requirements. This usually means for the architect to make decisions that balance the contradictory nature of different stakeholders requirements while keeping the total system costs under defined restrictions. It also means making use of these constraints and their related trade-offs as the basis of a common language used by stakeholders to efficiently help the architect to achieve his/her goal.

5. An architecture influences the organizational structure. The architecture of a system is the best description available of its top-down structures, then it is also a very effective start point for designing the organizational structures that will be in charge of task-breakdown.

6. An architecture is influenced by its environment. Environment elements such as quality attribute requirements, standard conformance,

organizational constraints, and professional context define a set of external conditions the architecture must function under.

7. An architecture documents the system. Every system has an architecture, whether officialy documented or not. However, properly documented architectures can function as an effective documentation for the system. Since an architecture captures the system's initial requirements, constraints, and stakeholder trade-offs, it is a good practice to document it properly.

8. An architecture often conforms to a *pattern*. Architectural patterns are sets of styles which have had a lot of success in practice. Most architectures conform to architectural patterns, like N-tier, Client-Server, or Pipes and Filters. In these cases, the job of the architect comes down to mixing and matching existing sets of such readily available patterns to solve the problem at hand. A body of work known collectively as architectural patterns or styles has catalogued a number of successfully used structures that facilitate certain kinds of component communication. These patterns are essentially reusable architectural blueprints that describe the structure and interaction between collections of participating components.

So far, a *software architecture* can be identified with the architecture a software system exhibits. This characterization leads to a set of specific features present in aspects like the system itself, its structure, the environment where it is situated and the stakeholders that have interest in it. Moreover, a software architecture can be further described by a set of characteristics that arise as concrete relations between the architecture and its key aspects.

2.3 Software Quality Attributes

According to [Gorton, 2006] a software architecture addresses the nonfunctional requirements of a system. Nonfunctional requirements can be classified in three areas: technical constraints, business constraints and quality attributes. Technical constraints refer to technologies such as operative system, programming languages, database systems, etc., that must be used because of the specific context of the system. Business constraints also refer to restrictions imposed by a particular context, but they originate in business considerations such as customer requirements or provider specifications. Technical and business constraints are usually nonnegotiable, and also depend on specific contexts that cannot be completely defined until a particular case is analysed. Therefore, a detailed description of technical and business constraints as nonfunctional requirements addressed by a software architecture is beyond the scope of this thesis. Software quality attributes, on the other hand, can be properly described as a set of characteristics the system must exhibit in order to satisfy issues of concern placed by its stakeholders. Following subsections are based on the work of [Gorton, 2006] and [Pillai, 2017], and describe some of the most relevant software quality attributes.

2.3.1 Performance

A performance quality requirement defines a metric that states the amount of work an application must perform in a given time, and/or deadlines that must be met for correct operation. Common metrics for performance measurement and evaluation are:

1. Throughput: the amount of work an application must perform in unit time.
2. Response Time: a measure of the latency an application exhibits in processing a business transaction.
3. Deadlines: a limited window of time given to an application to complete its task.

2.3.2 Scalability

Scalability is about how a design can cope with some aspect of the application's requirements increasing in size. Examples of such aspects are:

1. Request Load: the total number of requests placed to the application, regardless of the number of active clients.
2. Simultaneous Connections: the number of concurrent users served by the application.
3. Data Size: the amount of data processed by the application.

4. Deployment: the size of the user base the application has to be deployed and/or modified to.

2.3.3 Modifiability

The modifiability quality attribute is a measure of how easy it may be to change an application to cater for new functional and nonfunctional requirements. Designing likely change scenarios for the application as guides for future required modifications is a good practice to ensure an adequate level of modifiability for the architecture. Modifiability is also defined as the ease with which changes can be made to a system, and the flexibility at which the system adjusts to those changes. From an architect's perspective, the interests in modifiability is about difficulty, costs and risks associated with changes to the system at three levels: local, non-local and global.

2.3.4 Security

At the architectural level, security reduces to understanding the precise security requirements for an application, and devising mechanisms to support them. The most common security-related requirements are:

1. Authentication: Applications can verify the identity of their users and other applications with which they communicate.
2. Authorization: Authenticated users and applications have defined access rights to the resources of the system.
3. Encryption: The messages sent to/from the application are encrypted.
4. Integrity: This ensures the contents of a message are not altered in transit.
5. Nonrepudiation: The sender of a message has proof of delivery and the receiver is assured of the sender's identity. Therefore neither can subsequently refute their participation in the message exchange.

2.3.5 Availability

Availability refers to the property of readiness of a software system to carry out its operations when the need arises, and it is closely related to the reliability of the system and to its ability to recover from fails. As a result, important techniques ensuring availability are fault detection, fault recovery and fault prevention. The availability of an application is related to its reliability and, as a consequennce, to failures. Failures in applications cause them to be unavailable. Failures impact on an application's reliability, which is usually measured by the mean time between failures. The length of time any period of unavailability lasts is determined by the amount of time it takes to detect failure and restart the system. Consequently, applications that require high availability minimize or preferably eliminate single points of failure, and establish mechanisms that automatically detect failure and restart the failed components. Replicating application components is also a tested strategy for high availability.

2.3.6 Integration

Integration is concerned with the ease with which an application can be usefully incorporated into a broader application context. The value of an application or component can frequently be greatly increased if its functionality or data can be used in ways that the designer did not originally anticipate. The most widespread strategies for providing integration are through data integration or providing an API.

2.3.7 Portability

Portability is defined as the ability of an application to be easily executed on a different software/hardware platform to the one it has been developed for. It depends on the choices of software technology used to implement the application, and the characteristics of the platforms that it needs to execute on.

2.3.8 Testability

Testability refers to how much a software system is amenable to demonstrating its faults through testing, and can also be thought of as how much a software system hides its faults from end users and system integration tests. It is also related to how predictable a software system's behavior is: the more predictable a system, the more it allows for repeatable tests, and for developing standard test suites based on a set of input data or criteria. Testability is also related to how easy or difficult it is to test the application, and it is directly correlated to simplicity in design and to writing less original code by incorporating pre-tested components.

2.3.9 Supportability

Supportability is a measure of how easy an application is to support once it is deployed. Support typically involves diagnosing and fixing problems that occur during application use. Supportable systems tend to provide explicit facilities for diagnosis, such as application error logs that record the causes of failures. They are also built in a modular fashion so that code fixes can be deployed without severely inconveniencing application use.

2.3.10 Deployability

Deployability is the degree of ease with which software can be taken from the development to the production environment. It is more of a function of the technical environment, module structures, and programming runtime/languages used in building a system, and has nothing to do with the actual logic or code of the system. Some factors that determine deployability are the use of modular structures, the degree of similarity between development and production environments, the level of maturity of the development ecosystem, the use of standardized configurations and infrastructures and the use of containerization technologies.

Software quality attributes constitute a key concept for this research because they provide a practical way to express the system concerns, that is the concerns placed by the system stakeholders, in terms of definite architectural features such as performance, security, scalability, availability, etc. As it will be shown in next section, identifying the system stakeholders and their concerns is a fundamental element for any software architecture description or characterization.

2.4 The International Standard ISO / IEC / IEEE 42010

The ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description [ISO/IEC/IEEE, 2011] is an international standard for architecture description of systems and software. It has an antecedent in the IEEE Std 1471:2000, *Recommended Practice for Architectural Description of Software-Intensive Systems*, and was prepared by the Joint Technical Committee 1, *Information Technology*, integrated by the International Organization for Standardization and the International Electrotechnical Commision, in cooperation with the Software and Systems Engineering Standards Committee of the Computer Society of the Institute of Electrical and Electronic Engineers. Besides providing a formal specification for architecture description, the ISO/IEC/IEEE 42010 also specifies architecture viewpoints, architecture frameworks and architecture description languages. It gives motivations for terms and concepts used; presents guidance on specifying architecture viewpoints; and demonstrates its use with other standards as well.

2.4.1 Architecture versus Architecture Description

The ISO/IEC/IEEE 42010 specifies the manner in which architecture descriptions of systems are organized and expressed. It clearly distinguishes between architecture and architecture description: whereas an architecture is abstract, consisting of concepts and properties, an architecture description is a strictly defined *work product used to express an architecture*. The standard establishes there is no single characterization of what is essential or fundamental to a system, and that characterization could pertain to any or all of the following aspects:

- system constituents or elements;
- how system elements are arranged or interrelated;
- principles of the system's organization or design; and
- principles governing the evolution of the system over its life cycle.

On the other hand, the architecture description of a system is formally defined in full detail by the standard, and include the following contents:

1. Architecture description identification and supplementary information such as date of issue, status, authors, reviewers, approving authority, issuing organization, change history, summary, scope, context, glossary, version control information, configuration management information and references; as well as results from any evaluations of the architecture or its architecture description.

2. Identification of the system stakeholders and their concerns. Examples of applicable stakeholders are users, operators, acquirers, owners, suppliers, developers, builders, or maintainers; while examples of applicable concerns are the purposes of the system, the suitability of the architecture for achieving the system's purposes, the feasibility of constructing and deploying the system, the potential risk and impacts of the system to its stakeholders throughout its life cyle, and maintainability and evolvability of the system. Stakeholder concerns are directly related to nonfunctional requirements of the system and to software quality attributes previously discussed.

3. A definition for each architecture viewpoint used in the architecture description. An architecture viewpoint frames one or more concerns. It establishes the conventions for constructing, interpreting and analyzing a particular architecture view so the latter is able to address the concerns framed by the viewpoint. Viewpoint conventions can include languages, notations, model kinds, design rules, and/or modeling methods, analysis techniques, and other operations on views.

4. An architecture view and architecture models for each architecture viewpoint used. Views address one or more of the concerns placed by the stakeholders. A view is governed by the conventions established in the corresponding viewpoint and is composed of one or more architecture models. An architecture model uses modeling conventions appropiate to the concerns to be addressed and is governed by a model kind.

5. Applicable architecture description (AD) correspondence rules, AD correspondences and a record of known inconsistencies among the architecture description's required contents. Correspondences define relations between AD elements (stakeholders, concerns, viewpoints, views, model kinds, models, decisions, rationales) and can be governed by correspondence rules.

6. Rationales for architecture decisions. They record explanations, justifications or reasoning about architecture decisions that have been made.

The conceptual model of an architecture description as provided by the ISO/IEC/IEEE 42010 is shown in Figure 2.2. It also uses the conventions

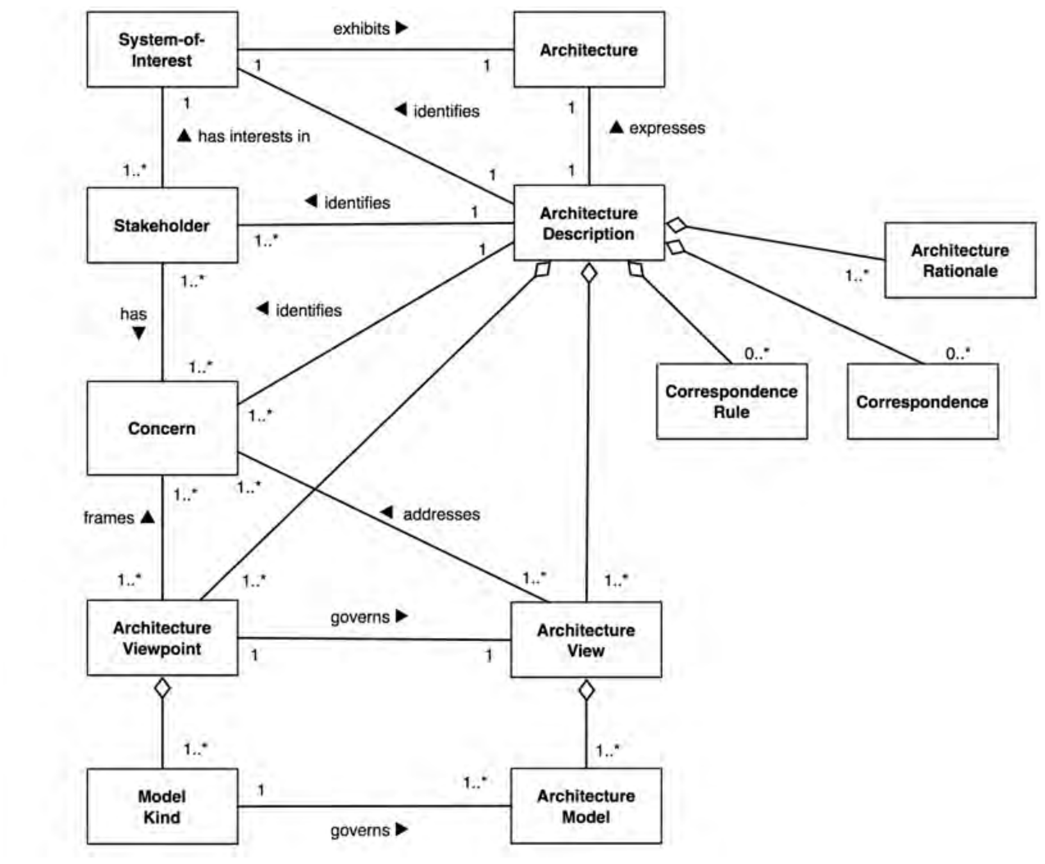

Figure 2.2: Conceptual model of an architecture description [ISO/IEC/IEEE, 2011]

for class diagrams defined in [ISO/IEC, 2005]. This highly detailed characterization of a system architecture depicts relations that are more specific than the ones reviewed in the architecture description context. Following are examples of such relations:

- A system-of-interest exhibits exactly one architecture, which is expressed by exactly one architecture description.
- An architecture description identifies exactly one system-of-interest, and also identifies one or more stakeholders that have interests in the system, as well as one or more concerns that system stakeholders have.

- An architecture description aggregates (diamond arrowhead) one or more architecture viewpoints, one or more architecture views, zero or more correspondence rules, zero or more correspondences, and one or more architecture rationales.
- An architecture view addresses one or more concerns, is governed by exactly one architecture viewpoint, and aggregates one or more architecture models.
- An architecture model is governed by exactly one model kind, but a model kind may govern one or more architecture models.

It is evident in the class diagram that an architecture description is a complex work product which involves the use of strict templates and even specialized description languages. For this reason, in order to build a consistent architecture characterization for ITSFCC within a suitable time frame, an alternative, reduced method was designed. This architecture characterization, proposed in next section, cannot be regarded as an architecture description, as it does not entirely comply with the specifications the ISO/IEC/IEEE 42010 defines. However it can be considered as a partial implementation of the standard, therefore as a basis for further efforts towards an ITSFCC architecture description.

2.5 Architecture Characterization for ITSFCC

The proposed architecture characterization for ITSFCC consists of the following elements:

1. Identification and overview.
2. Identification of stakeholders.
3. Identification of concerns in terms of software quality attributes.
4. Process-based view.
5. Block diagram-based architecture model.
6. Pattern-based configuration.

The following subsections briefly present each item of the suggested architecture characterization.

2.5.1 Identification and Overview

In compliance with the ISO/IEC/IEEE 42010, the first element of the characterization comprises the identification of the architecture and the following supplementary information: date of issuance, status, authors, issuing organization, change history, summary, context, glossary, and references. Part of the supplementary information is provided by this thesis document and the remaining part will be recorded as the revision history of the architecture.

2.5.2 Identification of Stakeholders

Stakeholders that may have interest in ITSFCC are primarily located in two areas: Intelligent Time Series Forecasting and Cloud Computing. While the former area is related to research domains like Time Series Analyisis, Data Science, Machine Learning, and Electric Power Industry, the latter is associated with topics like Networking, System Virtualization, Distributed Computing, and Service Ecosystems. As a consequence, stakeholders of ITSFCC face an interdisciplinary and complex domain when analysing the system's environment. For this reason primary stakeholders are the members of the Data Science Research Work Group (The Work Group) at the Graduate School of Electrical Engineering (DEPFIE), Universidad Michoacana de San Nicolás de Hidalgo (UMICH). Researchers and graduate students in the Work Group conform an academic team experienced enough to deal with the design, building, testing, and deployment of ITSFCC; consequently they act as development stakeholders with strong interests in the research possibilities of ITSFCC's functionality.

2.5.3 Identification of Concerns in Terms of Software Quality Attributes

Since primary stakeholders of ITSFCC have been identified as academic developers with strong interests in research activities, the concerns they place to the system must be expressed accordingly. Software quality attributes provide the basis for expressing these concerns, as long as no business or technical constraint has to be considered at this stage. Performance, scalability, security, availability, and supportability are quality attributes that result significant not only to current academic stakeholders, but also to potential system users in the future. They clearly impact system development and utilization and, on this ground, will be explicitly stated when addressing specific concerns in further chapters. On the other hand, modifiability, integration, portability, testability, and deployability will not be addressed, at least not explicitly, in architecture characterization.

2.5.4 Process-based View

Although no architecture viewpoint is defined in this characterization, an architecture view is still considered. It is based on the process view portrayed by [Kruchten, 1995] as part of the renowned *4+1 View Model of Architecture*. This process view is able to address quality attribute-based concerns, to focus on architecturally significant elements and to depict the high-level structure of the system. A process, as the key concept of this view, is considered as a group of tasks that: form an executable unit which can be tactically controlled (started, recovered, reconfigured, shut down, and so on), can be replicated to distribute processing load or improve system availability, and can be partitioned into a set of independent tasks. The process-based view considers processes as basic components of the system, while messages or calls are regarded as connections between processes. The system is consequently represented as a high-level structure made up of blocks, detailed or simplified, and connection lines. For this reason the only architecture model the view aggregates is governed by the block diagram model kind.

2.5.5 Block Diagram-based Architecture Model

According to the standard, a model kind captures conventions for a type of modeling, like languages, notations, techniques, analytical methods, etc. In an architecture description, each architecture model aggregated in an architecture view is governed by exactly one model kind. ITSFCC architecture characterization includes an architecture model that, in accordance with the aforementioned process-based view, is governed by the block diagram model kind. A block diagram represents a system as a set of blocks connected by lines. Blocks are principal parts or functions of the system while lines simbolize the relationships between blocks. Block diagrams are intended for high-level representation of complex systems, where blocks are commonly black boxes that just return outputs from defined inputs. This representation is adequate when details are not relevant for the analysis, or when they do not exist or have not been implemented yet.

2.5.6 Pattern-based Configuration

A last element of ITSFCC architecture characterization is a pattern-based configuration. As previously stated, patterns are sets of styles which have had a lot of success in practice, are reusable and readily available, and describe the structure and interaction between collections of participant components. Architectural patterns conform the preferred design strategy for this research work because they consolidate a strong body of collective knowledge that has been tested and refined through its application in many different environments by many architect teams. Moreover, a pattern-based configuration for ITSFCC is not only a tribute to the idea of *standing on the shoulders of giants*, but also the only possible way to accomplish this research's objectives within the scheduled time limits.

Summary

This chapter has shown the concept of software architecture based on the standard definition of system architecture and on a set of characteristics that arise as concrete relations between the architecture itself and its key aspects, when analysing a software system. The concept of architecture description was presented on the basis of the ISO/IEC/IEEE 42010 international standard. Differences between architecture, architecture description, and architecture characterization were also discussed. Finally, an architecture characterization for ITSFCC was proposed as a partial implementation of the international standard for architecture description. This characterization is intended to provide a methodological guide for the next two chapters, where architectural elements will be developed for addressing the particular environments related to Intelligent Time Series Forecasting and Cloud Computing.

Chapter 3

Architecture Elements for Intelligent Time Series Forecasting

This chapter introduces the concept of intelligent time series forecasting and examines the architectural elements a software application requires in order to address it. In accordance to the architecture characterization developed in previous chapter, Section 3.1 describes the system environment in terms of the high complexity that characterizes time series forecasting for the energy domain and the Artificial Intelligence and Machine Learning-based methods developed to cope with it. Section 3.2 identifies concerns placed to ITSFCC by its academic stakeholders and expresses them as broad features the system must include. Features are then mapped to application components that can be related to precise software quality attributes. Finally, Section 3.3 presents software pattern formats and describes the architectural patterns suggested to deal with the identified concerns.

3.1 Description of System Environment

Although there is no formal definition for the term *intelligent time series forecasting*, it is used in this work to encompass the broad use of Artificial Intelligence and Machine Learning-based methods of time series forecasting, with a special focus on the energy domain. As defined, intelligent time series forecasting is a fundamental research line for the Work Group. In order to delimit ITSFCC environment, two research domains must be briefly described: the complexity involved in time series forecasting for the energy domain, and the intelligent-based methods that have been developed as a resource to deal with such high complexity. The following two subsections address these topics.

3.1.1 Complexity of Time Series Forecasting in the Energy Domain

Time series forecasting in the energy domain conforms an extensive network of activities spreaded across the fields of generation, distribution, transmission, and demand. This subsection is primarily based on the work of [Dannecker, 2015] and its objective is to present diverse situations commonly found in the forecasting network that operate as factors of a significant complexity. Following is a list of such factors, then a brief discussion of them in subsequent paragraphs:

- Multiple forecast scopes and horizons.
- Forecast data characteristics.
- Context-aware forecast models.
- Multi-component forecast models.
- Hierarchical data warehouses.
- Renewable energy sources.
- Smart grids.

Multiple Forecast Scopes and Horizons

Forecasting in the energy domain encompasses many different processes with particular scopes that require accurate predictions in multiple horizons. This broad landscape can be summarized in terms of horizon spans and examples of associated tasks as follows:

- Very short term (seconds to minutes): turbine control.

- Short term (minutes and hours up to day-ahead): power plant operations, grid balancing and scheduling, real-time unit dispatching, automatic generation control, operation reserve planning and control, real-time electricity market trading and administration, peak load analysis, load following, etc.
- Medium term (days): pre-dispatch, unit commitment, day ahead trading, and maintenance planning, etc.
- Long term (weeks): maintenance planning, improving balance area control, system planning, investment planning, etc.

Forecast Data Characteristics

Complexity in time series forecasting for the energy domain is also due to a diversity of characteristics observed in time series specific to this domain, that must be considered by forecast models, like:

1. Seasonal Patterns. Aggregated energy demand commonly exhibits three different seasonal components related to daily, weekly, and yearly consumption patterns.
2. Aggregation-Level-Dependent Predictability. In general, the more entities, like electricity meters, plants, cities, regions, etc., are aggregated in the forecasting process, the better the predictability of forecasts.
3. Time Series Context and Context Drifts. When it comes to forecasting for the energy domain, the temporal development of time series is driven by a conglomeration of background processes and external influences, such as working calendar, meteorological behavior, and economic factors. Usually this *context* is not static but presents dynamics or *drifts* that are observable and measurable, and can be considered as valuable information for enhancing predictability.

Context-aware Forecast Models

Most of the time series context in forecasting for the energy domain (calendar, meteorological or economic-driven) is observable. Thus, a better forecast can be achieved if additional time series describing this context are incorporated into the model. As a consequence, complexity increases because variables in context time series raise the dimensionality of models. For instance, [Tam and Sehgal, 2014] propose a Forecasting-as-a-Service framework intended to provide on-demand forecasts of solar or wind power at locations specified by the customers. A fundamental part of this framework consists of a large volume of data pertinent to renewable energy forecasting such as weather and geographic information, measurement data, and equipment specifications. This data is collected over the internet and comes from a variety of sources like federal agencies, national databases, private organizations, universities, equipment vendors, etc.

Multi-component Forecast Models

A useful approach to develop a very accurate forecast model is to split the data into logical subseries and to produce a separate model for each one of them [Dannecker, 2015], [Rangel et al., 2017], [Lopez Farias et al., 2018]. For instance, instead of a single-equation ARIMA model built upon a singular time series of load values, a 24-equation ARIMA can be generated on the basis of registered load values for each hour of the day. The same applies for models that are not based on equations, like Artificial Neural Networks or Support Vector Machines, then a single-component model is transformed into a multi-component one. Complexity increases under this approach because numerous additional parameters are required to be optimized.

Hierarchical Data Warehouses

Forecasting in the energy domain encompasses a hierarchical structure made up of diverse entities such as devices, meters, final users, distribution, transmision or generation plants, over cities, states, countries or regions. Forecasting in a hierarchical environment typically means that higher level entities forecasts are based on the aggregated data of lower level entities. Even though this hierarchical operation simplifies model production at higher level entities, required data synchronization over associated data warehouses implies additional efforts from energy management systems.

Renewable Energy Sources

Forecast of future energy consumption and production is a fundamental requirement for the stability and the day to day operation on the electricity grid. While demand forecasts are well-established since years, the increasing amount of renewable energy sources substantiate the need to also provide accurate forecasting for renewable supply. The reason is that renewable energy sources cannot be planned or dispatched like conventional energy sources, since they strongly depend on environmental conditions and most importantly on the current state of weather. Once again forecasting complexity increases as long as weather prediction must deal with nonlinear, chaotic and/or noisy data.

Smart Grids

In present time, extensive use of Information and Communication Technology (ICT) across electricity production, distribution, and consumption simultaneously enriches and complicates energy management activities, including forecasting. For instance, Internet of Things (Internet of Things) devices such as smart meters or smart home systems and appliances make it possible to accomplish demand side management, providing increased flexibility to energy management systems. Smart grids make use of this flexibility to deal with recurrent modifications to energy production schedules caused by intermitent availability of renewable energy sources. In other words, flexibility provided by smart grids comes at the cost of adding numerous layers of information to the forecasting network, rising its complexity to a Big Data scale [Daki et al., 2017].

All of the aforementioned factors make time series forecasting for the energy domain a complex process that demands the application of efficient and reliable methods. Artificial Intelligence and Machine Learning-based methods, briefly addressed in the next subsection, constitute an important approach to deal with this complexity.

3.1.2 Intelligent Methods for Time Series Forecasting

Time series forecasting for the energy domain involves multiple tasks that must be performed at the different levels of an extensive, hierarchical information structure. It must also consider the diverse relationships information levels tend to establish among them. This complex landscape has motivated the development of multiple methods to accomplish the fundamental goal of producing accurate and efficient predictions. This subsection presents a couple of classifications for time series forecasting in the energy domain where Artificial Intelligence and Machine Learning-based methods outstand as novel approaches to address complexity. In addition to produce accurate and fast solutions, intelligent forecasting methods are suitable to merge into hybrid approaches that combine specific strengths from individual methods in order to enhance predictability [Flores et al., 2009], [Hong, 2013], [Rodriguez et al., 2016], [Rangel et al., 2017]. Although a complete review of the different methods applied to time series forecasting is beyond the scope of this work, the reader is advised to examine, if required, the sources of the two cited classifications.

Classification of Time Series Forecasting Methods

A classification of forecast models for the energy domain with three categories is provided in [Dannecker, 2015]. It includes autoregressive models, exponential smoothing models, and Machine Learning-based models. This source identifies Bayesian Networks, Artificial Neural Networks, and Support Vector Machines as the most important examples for Machine Learning techniques used for forecasting. It also states that the results in several research studies that evaluated the use of Machine Learning approaches for forecasting energy supply and demand are controversial, and that the advantage of certain modeling technique over the others highly depends on the use-case, in this context the specific sequence of interactions between the user and the system, and the existing data.

Focusing on a more specific domain [Hong, 2013] proposes a classification for electric load forecasting methods based on their primary technological development, as follows:

1. Traditional approaches or mathematical relationship models, including Box-Jenkins Auto Regressive Integrated Moving Average (ARIMA), Auto Regressive Integrated Moving Average with Exogenous Input (ARIMAX), Seasonal Auto Regressive Integrated Moving Average (SARIMA), exponential smoothing models, including Holt-Winters (HW) and Seasonal Holt-Winters' linear exponential smoothing (SHW), state space/Kalman filtering, and linear regression.

2. Artificial Intelligence-based approaches, including Knowledge-Based Expert System (KBES), artificial nerural networks (ANNs), and fuzzy inference system.

3. Hybrid approaches that integrate traditional and AI-based models, such as the support vector regression (SRV) and its related hybrid/-combined models.

This classification groups Box-Jenkins and Holt-Winters models with their variants into a single category, and places intelligent forecasting methods in a second one. By allocating all combined forecast methods in a third category, it underlines the importance of the numerous efforts that have been conducted towards enhancing model predictability by hybridizing basic forecasting approaches. An exhaustive review of recent research on electric load forecasting is given by the preceding source, including references to several cases of hybrid operation. Moreover, it proposes an Electric Load Forecasting Support System (ELFSS) which includes clear hybridization rules and processes. This ELFSS framework will be briefly described in the following paragraphs, as an example of a hybrid operation model of forecasting for the energy domain that helps to identify specific architectural requirements for this application field.

Hybrid Operation in Energy Demand Forecasting

Hong's ELFSS is a valuable framework to understand extensive hybrid processes in energy demand forecasting. ELFSS is depicted in Figure 3.1 and can be summarized as follows [Hong, 2013]:

- It is based on a Support Vector Regression (SRV) forecast model. SRV has been successfully employed to solve forecasting problems in many fields, such as financial time series forecasting, production value forecasting of machinery industry, software reliability forecasting, atmospheric science forecasting, tourism forecasting, etc.

- It deploys evolutionary algorithtms, such as Genetic Algorithm (GA), Simulated Annealing (SA), Immune Systems (IA), Particle Swarm Optimization (PSO), Tabu Search (TA), and Ant Colony Optimization (ACO) as optimization techniques to obtain the best parameter combination for the SRV model.

- It employs fuzzy logic to construct an inference system to preprocess the time series data and find out or define the characteristic rule sets of data pattern, such as linear, logarithmic, inverse, quadratic, cubic, compound, power, growth, and exponential. Pre-processed data is then associated to the appropriate pattern in a three-value classification: fluctuation, regular, or noise.

- Filtered data is then passed to the SRV model. The evolutionary algorithm used for parameter optimization depends on the data pattern previously found: according to experimental results, SA and ACO perform better for fluctuation or noise patterns, while GA, TA, IA, and PSO are selected in presence of a regular pattern.
- In addition, to avoid getting trapped in local minima, suitable chaos or cloud theory and appropriate (recurrent or seasonal) mechanism could be further hybridized or combined with associated evolutionary algorithms. These mechanisms lead to the deployment of Chaos/Cloud variants of original evolutionary algorithms: CGA, CSA, CTA, CIA, CACO, and CPSO.
- Finally, ARIMA, exponential smoothing, and regression forecast models are built and run on filtered data in order to obtain benchmarking references for the hybrid SRV-(Chaos/Cloud) Evolutionary Algorithm model.

The ELFSS framework illustrates the extensive hybrid operation of contemporary time series forecasting systems, which offer a way to overcome specific drawbacks identified in singular forecasting techniques by combining the particular strengths of different methods into a more effective mixed solution. As long as hybridization is strongly based on Machine Learning and Artificial Intelligence techniques, like support vector regression, evolutionary algorithms, fuzzy logic, artificial neural networks, deep learning, etc., it can be regarded as an important feature of intelligent time series forecasting.

In addition to ELFSS, a more concrete and specific hybrid approach worth to review is the Nearest Neighbors - Differential Evolution (NNDE) system, developed inside the Work Group by [Flores et al., 2017]. Contents in the remainder of this section are entirely based on this source.

NNDE as a Representative Model for ITSFCC

NNDE is a forecasting method currently at an advanced development stage inside the Work Group and, hence, it is well-known to ITSFCC stakeholders. Similar to the ELFSS framework, it is Machine Learning-based and

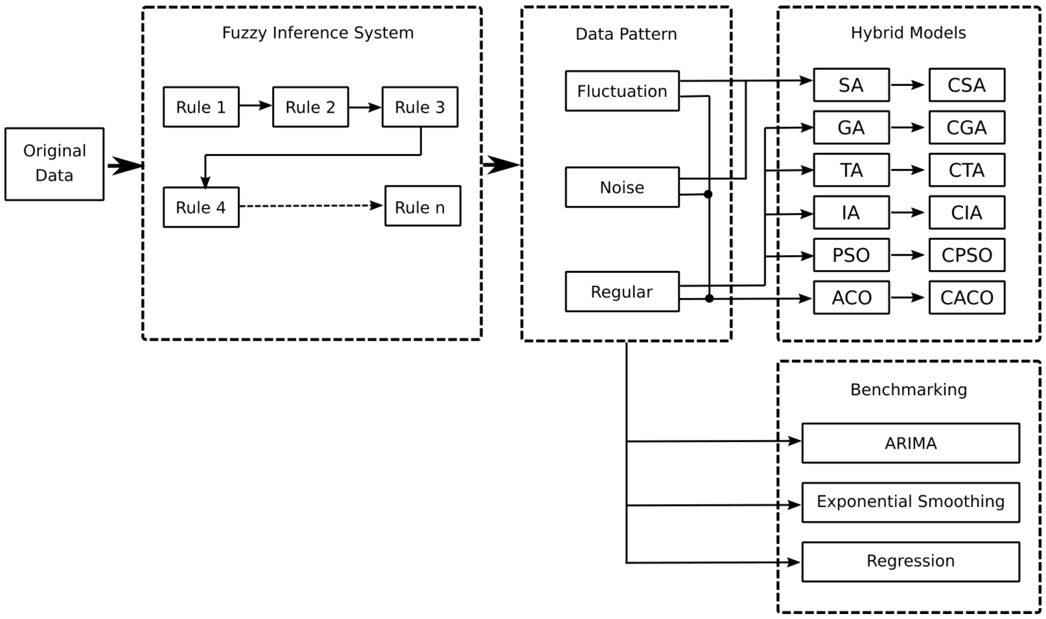

Figure 3.1: Hong's Electric Load Forecasting Support System (based on [Hong, 2013])

presents a fundamental hybrid operation, then it can be used as a representative forecast model for ITSFCC. Finally, for the reason that it resides in the Work Group, it provides a close and precise guide for system concerns to be addressed in ITSFCC architectural design and characterization. NNDE is based on the Nearest Neighbors (NN) algorithm [Kantz and Schreiber, 2004], which has been successful in different areas of pattern recognition. NN searches for similar instances recovering them from a large dimensional feature space and incomplete data. It assumes that sub-sequences of a time series that emerged in the past are likely to have a resemblance to the future sub-sequences and can be used for generating NN-based forecasts. NN offers simple coding, as well as efficient and fast execution, however it is sensitive to changes in the input parameters (i.e. the length of the radius of the neighborhood, the embedding dimension, and the delay between measurements). Those parameters must be adequately optimized in order to obtain accurate results. Parameter optimization is based on Differential Evolution (DE) [Price et al., 2006], an evolutionary algorithm that has recently proven to be a valuable method for optimizing real valued multi-modal objective functions. It is a parallel direct search method having good convergence properties and simplicity in implementations. Furthermore, DE has recently motivated the development of parallel schemes to execute the optimization algorithm on distributed environments, such as Apache Spark. DE executed in parallel results in significant advantages on performance and scalability, when compared to conventional serial implementations [Teijeiro et al., 2016].

In summary, NNDE can be regarded as an intelligent time series forecasting method that exhibits a fundamental hybrid principle. Consequently, it can be employed as a representative model for ITSFCC, which means it can be referenced not only as the first model to be included in the system, but also as a good approximation to design and development challenges placed by other intelligent forecasting methods to be added in the future. In combination with the ELFSS framework, NNDE can be used as a target scenery, a representative source to elaborate ITSFCC concerns. Therefore, NNDE can be considered as a design guide for the system's architecture characterization. Next section is precisely devoted to the identification of ITSFCC concerns by summarizing the software quality attributes required to deal with the complexity of time series forecasting using intelligent methods.

3.2 Identification of System Concerns

As stated in the previous chapter, concerns from ITSFCC's stakeholders must be translated to non-functional requirements, specifically to software quality attributes, as long as no technical or business constraint has to be considered at this stage of the ITSFCC design. In order to achieve this objective, a set of broad features required by the system to face the problems located in the environment will be listed. Broad features will then be mapped to several application components which offer two important advantages:

1. They can be clearly associated, individually or as a whole, to concrete software quality attributes.

2. They provide a basis to identify the architecturally significant elements of the process view required by the architecture characterization.

3.2.1 Definition of Software Quality Attributes

The collection of values assigned to ITSFCC's quality attributes primarily depend on the expected work load the Work Group will place to the system under normal operation. Presently, the Work Group comprises aproximately 10 researchers on 5 facilites, located in 5 different states of Mexico. The work load for ITSFCC will be intermitent and activated on request, not on schedule. The system will have to be available for the 99% of total operation requests. The target response time for complex forecasting operations will be initially set to 60 minutes. Finally, the target time for the system to be fully functional after scheduled shutdown or failure will be initially set to 10 minutes. Consequently, the initial collection of values for ITSFCC's quality attributes are:

- Availability: ITSFCC will have to be available for 99% of total operation requests.
- Scalability: work load from 10 to a peak of 20 users, located in 5 to 10 different cities, working simultaneously in forecast model production and evaluation.
- Performance: initial forecasting results must be delivered in less than one hour of operation.
- Supportability: after unavailability periods due to failure or scheduled maintenance, ITSFCC will have to be fully functional again in less than 10 minutes.

The remainder of this section identifies ITSFCC concerns using each broad feature name as subsection header. The subsection body is conformed by a simple description of the feature and an item list including its associated application components. Software quality attributes are finally summarized in the last subsection. Whenever a quality attribute is exclusively related to a specific application component, it is identified within the item content. According to the ITSFCC architecture characterization, this process involves only performance, availability, scalability, security, and supportability, because these quality attributes are significant not only to current academic stakeholders, but also to potential stakeholders in the future, like system users, customers, marketing team, suppliers, etc.

3.2.2 Complexity Management

Complexity in time series forecasting for the energy domain is the result of multiple factors that demand the management of numerous time series, which exhibit multiple relationships and hierarchies among them, and are likely to grow up to a Big Data scale.

- Time Series Repository: A number of time series, in the order of hundreds, each one of them composed of a number of elements accountable in millions when considering its length and dimensionality, must be stored in a repository able to provide required speed for computing operations, high availability in case of disk, file system or database failures, and efficient metadata management. Availability: 0.99 during on-request operation. Scalability: Peak load of 20 concurrent forecast processes.

- Batch Processing: All of the aforementioned time series must be efficiently processed in batch mode with results available in sub-one hour intervals. Processing tasks must consider the possibility of prior *subsequencing* of the time series, so they can be treated as a single sequence (a time series of daily values), or as a set of sequences obtained by regularly splitting the main sequence (24 time series of hourly values). Availability and Scalability: as previously stated. Performance: sub-one hour intervals for task completion.

- Graph Representation: Operations performed on time series must be represented and stored as graphs, where nodes identify input and resulting time series while edges identify operators. This representation will provide a flexible method to script complex operation sequences, produce variations in intermediate results, and rollback to previous processing stages when needed. Fig. 3.2 shows an example of this graph representation of time series operations. A series identified as 01 is clipped under a threshold of 4.5 on the axis 0 and the resulting series is identified as 02 and stored in the serialized file 02.pkl. A series identified as 03 comes as the result of an alternative operation at a threshold value of 3.5. A couple of series identified as 04 and 05 come as the result of applying a difference operation on series 03, at lag values of 1 and 2, respectively. All series are stored as serialized files referenced by the output_dataset property of the edge that identifies the operation. Availability and Scalability: as previously stated.

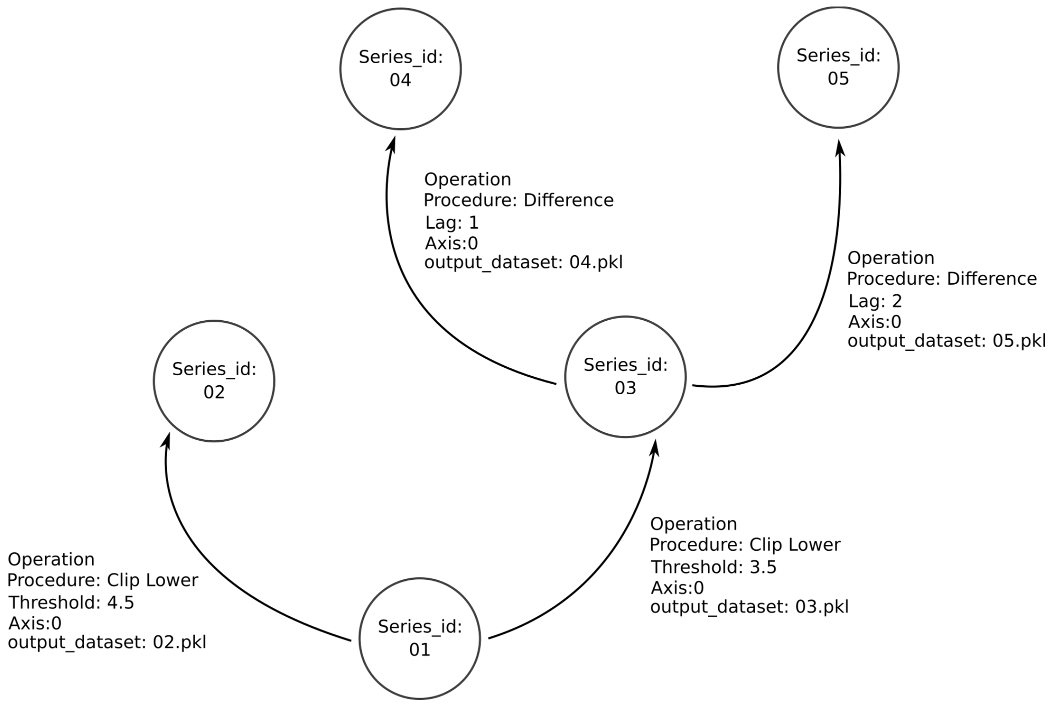

Figure 3.2: Example of graph representation of time series operations.

3.2.3 Hybrid-operation Forecasting and Benchmarking

ITSFCC's hybrid operation demands a reliable and efficient communication channel between the programs that support the different functionalities of the system, for instance the Nearest Neighbors algorithm and the Differential Evolution optimizer integrated inside the NNDE approach. This also applies to the different programs that support benchmarking operations, where the results of intelligent forecasting must be compared to conventional methods such as autorregresive and exponential smoothing models.

- Publication-Subscription Channel: Results obtained from a certain program must be broadcasted as messages into appropriate queues, where they can be asynchronously extracted by other programs that require these results in order to continue their execution. This *pub-sub channel* must allow different applications to exchange information in a reliable way, regardless of the programming languages they are written in, or the data formats they use. Availability, Scalability and Performance: as previously stated. Supportability: Component must be ready from shutdown to fully functional state in sub-ten minute periods.

3.2.4 Context-aware Forecasting

Context-aware forecasting places two additional requirements to ITSFCC: the procurement of context-related time series from multiple information sources, and the ability to keep previously generated forecast models along with their context-related time series, for reference purposes.

- Multi-source Data Extraction: The system must be able to acquire data from multiple sources over the internet to conform a detailed context for the time series used to produce the forecast model. This includes calendar, meteorological and economic-driven data, comming from many different repositories, where they might be stored in multiple formats. Availability and Scalability: as previously stated. Security: All extraction operations must be performed over secure connections.

- Forecast Model Repository: As proposed in the work of [Dannecker, 2015], for any given time series the system must be able to (1) keep previously generated forecast models along with their context information, (2) evaluate the similarity between a present context and contexts in history, and (3) use this context similarity as a criterion for a fast selection of the best forecast model in the repository. This model can then be used to provide almost immediate, preliminary predictions, while waiting for the newest model formulation. Besides context similarity, other criteria suggested to be considered are time series shape similarity, accuracy of the historic forecast model, and the elapsed time since that forecast model was last used. Availability and Scalability: as previously stated.

3.2.5 Efficient Model Production and Parameter Optimization

Complexity management, hybrid operation and the consideration of diverse context-related datasets for intelligent time series forecasting are likely to stress the ITSFCC processing components to a great extent, especially during parameter optimization operations. This situation places a critical requirement for the system: the ability to configure and execute parallel computing processes over a distributed platform.

- Parallel Implementation: Whenever possible, the system must implement parallel processing components able to produce a forecast model and to find its optimal parameters in sub-one hour periods. Regarding NNDE as a representative model for ITSFCC, parallel processing can be implemented, via Apache Spark, to speed up the Nearest Neighbors algorithm [Maillo et al., 2017] or the Differential Evolution optimizer [Teijeiro et al., 2016]. In addition, Apache Spark has been suggested to be used as a unified cluster computing platform, suitable for storing and performing Big Data analytics on smart grid data applications, both in batch processing and real-time mode [Shyam et al., 2015]. Availability, Scalability, Performance and Supportability: as previously stated.

3.2.6 Summary of System Concerns

Fig. 3.3 represents a summary of the system concerns identified in this section. It shows broad features required by ITSFCC to solve problems located in the environment, associated application components, and related software quality attributes. Notice that the data management components primarily call for availability and scalability, while Multi-source Data Extraction additionally calls for security during connections required to extract datasets. Processing and messaging components call for availability, scalability, and also for performance and supportability. Performance can be achieved by setting high computing specifications for server instances, while supportability can be enforced via analytics processes over failure log records and automated infrastructure. Next section will present the software patterns suggested to deal with the identified system concerns for intelligent time series forecasting operations.

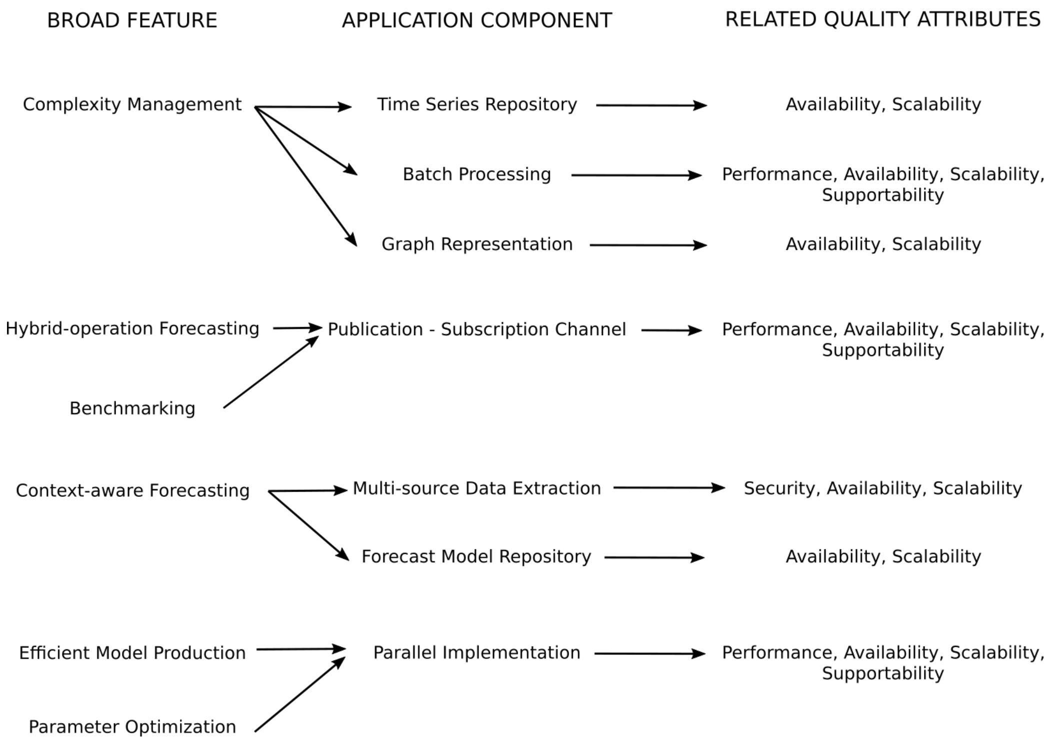
BROAD FEATURE
APPLICATION COMPONENT
RELATED QUALITY ATTRIBUTES
Complexity Management
Time Series Repository
Availability, Scalability
Batch Processing
Performance, Availability, Scalability, Supportability
Graph Representation
Availability, Scalability
Hybrid-operation Forecasting
Benchmarking
Publication - Subscription Channel
Performance, Availability, Scalability, Supportability
Context-aware Forecasting
Multi-source Data Extraction
Security, Availability, Scalability
Forecast Model Repository
Availability, Scalability
Efficient Model Production
Parameter Optimization
Parallel Implementation
Performance, Availability, Scalability, Supportability

Figure 3.3: Summary of system concerns related to Intelligent Time Series Forecasting

3.3 Software Patterns Related to System Concerns

Once the system concerns derived from intelligent time series forecasting have been identified and summarized as a set of application components related to relevant software quality attributes, the next step is to describe the software patterns that can be applied in order to deal with the context. This description requires an adequate software pattern format.

3.3.1 Software Pattern Formats

Different formats have been used in literature for software pattern specification. Suitable examples can be found in [Doddavula et al., 2013b], [Fehling et al., 2014], and [Raj et al., 2017]. These formats share essential characteristics and tend to differ only in minor features. The pattern specification forward used in this work comes from [Fehling et al., 2014], which is the main pattern source, and comprises the following attributes:

- Name: used to identify the pattern.
- Intent: short statement of the purpose and goal of the pattern.
- Driving Question: captures the problem that is answered by the pattern.
- Icon: graphical representation of the pattern.
- Context: describes the environment and forces leading to the problem solved by the pattern.
- Solution: briefly states how the pattern solves the problem raised by the driving question.
- Result: in this part the solution is elaborated in greater detail. The behavior of the application after implementation of the pattern is also discussed.
- Variations: covers slightly different forms of application of the pattern.
- Related Patterns: describes interrelations of the pattern with others, such as dependencies or exclusions.
- Known Uses: Covers existing applications implementing the pattern, as well as products that offer the pattern or support its implementation.

This format was selected because it provides a complete and useful characterization of patterns focused on cloud computing, which constitutes a major directive for ITSFCC. This collection of patterns is available, along with valuable graphic resources, not only in the main pattern source but also on the Internet, at the domain name clearly indicated in the introduction of the book. For this reason, patterns presented in this section will only display its *name* in the topic header, followed by its *intent* and the description of a specific *context* associated to the ITSFCC design. If required by the reader, standard context for the pattern, as well as the remaining attributes, can be reviewed directly on the main pattern source. Whenever a pattern presented in this section does not come from the main pattern source the alternative source will be explicitly cited. Finally, patterns are aggregated by subsection following the classification provided in the main pattern source.

3.3.2 Cloud Offering Patterns

Patterns in this subsection correspond to virtual resources that are offered via a self-service interface over a network.

Blob Storage

Intent: Data is provided in form of large files, or Binary Large OBject files, that are made available in a file system-like fashion by storage offerings that provide elasticity.

ITSFCC Context: This pattern is intended to provide backup and active storage for large objects that must be organized and made available over a network for the system operation: serialized time series in the Time Series Repository, database snapshots from Graph Representation, log files from the Publication-Subscription Channel, datasets obtained from Multi-source Data Extraction, and models and contexts integrated in the Forecast Model Repository.

Relational Database

Intent: Data is structured according to a schema that is enforced during data manipulation and enables expressive queries of handled data.

ITSFCC Context: This pattern is intended to provide efficient metadata management for the Time Series Repository and the Forecast Model Repository, as well as basic analytics and replication services for essential structured data obtained from Multi-source Data Extraction.

Key Value Storage

Intent: Semi-structured or unstructured data is stored with limited querying support but high-performance, availability, and flexibility.

ITSFCC Context: This pattern is intended to provide high-performance active storage for nodes and edges in Graph Representation, as well as for analytics of log files resulting from the Publication-Subscription Channel. It also provides replication services for essential unstructured or semi-structured data obtained from Multi-source Data Extraction.

Eventual Consistency

Intent: If data is stored at different locations (replicas) to improve response time and avoid data loss in case of failures, performance and the availability of data in case of network partitioning are enabled by ensuring data consistency eventually and not at all times.

ITSFCC Context: This pattern is intended to provide increased availablity and performance in case of network partitioning to the replicated databases in the Time Series Repository, the Forecast Model Repository, and the Graph Representation components.

Map Reduce

Intent: Large data sets to be processed are divided (mapped) into smaller data chunks and distributed among processing application components. Individual results are later consolidated (reduced).

ITSFCC Context: This pattern is intended to provide a basis for high-performance processing components in Batch Processing and Parallel Implementation.

Block Storage

Intent: Centralized storage is integrated into servers as a local hard drive managed by the operating system to enable access to this storage via the local file system.

ITSFCC Context: This pattern is intended to provide fast and reliable access to essential data for all the application components, in the form of centralized data buffers located between Blob Storage and processing stages. It also facilitates automatic infrastructure and platform deployment via scripting operations that are stored on and executed from it.

Message-Oriented Middleware

Intent: Asynchronous message-based communication is provided while hiding complexity resulting from addressing, routing, or data formats from communication partners to make interaction robust and flexible.

ITSFCC Context: This pattern is intended to provide a basis for the operation of the Publication - Subscription Channel.

3.3.3 Cloud Application Architecture Patterns

Patterns in this subsection correspond to architectural patterns that describe how applications have to be designed to benefit from a cloud environment.

Data Lake

Intent: According to [Raj et al., 2017] the data lake architecture pattern provides efficient ways to achieve reusing most of the data infrastructure and, at the same time, get the benefits of big data paradigm shifts. Data lakes have the following essential characteristics to address: manage abundant unprocessed data, retain data as long as possible, ability to manage data transformation, and support dynamic schema.

ITSFCC Context: This pattern is intended to provide efficient mechanisms for ingestion, storing, and conditioning of data obtained from Multisource Data Extraction.

Machine Learning

Intent: According to [Raj et al., 2017] this pattern helps to find a pattern of data inputs generated from heterogeneus devices such as RFID devices, energy meters, signal devices, weather-related devices, etc. Understanding data generated by automated systems or devices without manual intervention is a challenging task that requires to rely on algorithms and statistical methods.

ITSFCC Context: This pattern is intended to provide a basis for the use of algorithms like Nearest Neighbors in NNDE, Artificial Neural Networks, Support Vector Machines, etc., for Batch Processing and Parallel Implementation. It includes special data structures and types required for optimal ITSFCC processing, such as the tensor object from the TensorFlow library.

Big Data Analytics

Intent: According to [Doddavula et al., 2013a] the intent of this pattern is to provide a low-cost, large-scale analytics solution on Big Data for scenarios like fraud detection, product recommentations and Enterprise Data Warehouse

ITSFCC Context: This pattern is intended to provide a basis for the effective integration of all the application components, in compliance with low-cost, large-scale and high-performance directives.

Summary

This chapter introduced the concept of *intelligent time series forecasting* as an important element of the foundation for ITSFCC environment. It also presented the system concerns which were summarized as a set of broad features the system must exhibit, a set of application components that support those features, and a set of precise software quality attributes related to application components. Software pattern formats were introduced, and then used to describe architectural elements that are advised to deploy in order to cope with ITSFCC concerns. Next chapter will approach this analysis from the point of view of the second fundamental research domain involved in ITSFCC: *cloud computing*.

Chapter 4

Architecture Elements for Cloud Computing

This chapter presents the concept of *cloud computing* and examines the basic architectural elements an intelligent time series forecasting application must implement to exploit cloud computing potentials. According to the architecture characterization developed in Chapter 2, Section 4.1 describes the system environment in terms of the standard definition of cloud computing and the technological and organizational prerequisites for its adequate implementation. Section 4.2 identifies concerns related to properly leverage cloud computing potentials in ITSFCC design. The system concerns are expressed as a set of broad features that must be included, the application components required to achieve such features, and a set of precise quality attributes linked to components. Finally, Section 4.3 describes the software patterns suggested to deal with the identified concerns.

4.1 Description of System Environment

Regarding ITSFCC design process, there is a reason to separate the analysis of architectural elements related to *intelligent time series forecasting* from those correspondent to *cloud computing*: whereas the former concept refers to a clearly defined research domain, the latter points to a technology model which can be effectively applied for solving problems in many different domains. From the internal perspective of this work, intelligent time series forecasting places an extensive but nevertheless concrete problem to solve (vertical domain), while cloud computing offers a technical and methodological approach to the solution (transversal domain). Consequently, the description of ITSFCC environment related to these two topics is also different: while the previous chapter primarily dealt with problematic situations to face, this chapter alludes to potential opportunities found in cloud computing and the way to properly leverage them.

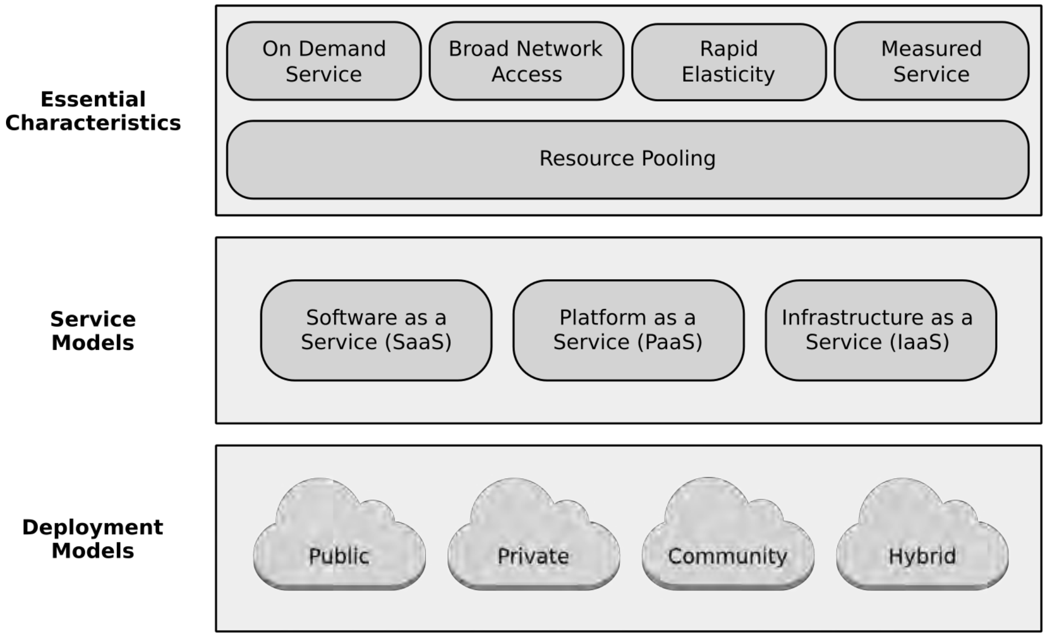

Figure 4.1: Cloud computing characteristics, service, and deployment models

4.1.1 Definition of Cloud Computing

The National Institute of Standards and Technology (NIST) provides the official definition of cloud computing in [Mell et al., 2011]: *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.* The elements in the definition are depicted in

Figure 4.1. Following are their complete descriptions as extracted from the aforementioned source.

Cloud Essential Characteristics

1. *On-demand self-service.* A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

2. *Broad network access.* Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms[1] (e.g., mobile phones, tablets, laptops, and workstations).

3. *Resource pooling.* The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

4. *Rapid elasticity.* Capabilities can be elastically[2] provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

5. *Measured service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability[3] at some level of ab-

[1]A thin client platform is a lightweight computer that basically monitors and controls processes executed in the server side, so it can work efficiently without advanced processing or storage components. On the other hand, a thick client platform usually performs most of the processing workload over data stored in the server side.

[2]In this context elasticity means the ability to flexibly and rapidly increase or decrease the quantity of IT resources to adjust them to the experienced workload.

[3]Typically this is done on a pay-per-use or charge-per-use basis.

straction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Cloud Service Models

1. *Software as a Service (SaaS).* The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure[4]. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

2. *Platform as a Service (PaaS).* The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider[5]. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

3. *Infrastructure as a Service (IaaS).* The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud

[4]A cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of cloud computing. The cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided, and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics. Conceptually the abstraction layer sits above the physical layer.

[5]This capability does not necessarily preclude the use of compatible programming languages, libraries, services, and tools from other sources.

infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Cloud Deployment Models

1. *Private cloud.* The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

2. *Community cloud.* The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

3. *Public cloud.* The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

4. *Hybrid cloud.* The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting[6] for load balancing between clouds).

4.1.2 Cloud Computing Fundamentals

The cloud characteristics, service and deployment models stated in the NIST definition point to important technical benefits that can be leveraged by IT departments. Among those benefits [Varia, 2010] identifies automatic and

[6]Cloud bursting refers to the ability to get infrastructure at runtime from a cloud provider and enabling the applications to use that to meet the service levels during a temporary spike in the workload [Doddavula et al., 2013b].

proactive scaling, a more efficient development lifecycle, improved testability, business continuity, and overflowing traffic to the cloud. However, in order to properly develop a cloud computing-based project it is essential to understand the technological and organizational fundamentals that support the model.

Technological Prerequisites

On the technological side, cloud computing depends on three complementary elements: virtualization, service-oriented architectures, and web services. These features can be summarized from [Baun et al., 2011] as follows:

- Virtualization. It allows an abstract, logical view on resources like servers, operating systems, data stores, and networks. Customers make use of this abstract layer by requesting virtual resources as services, incurring in much lower costs and efforts than those required for deployment and configuration of physical resources. Virtualization provides IT departments with important benefits such as dynamic scalablility, high availability, and simple, clear access to resource pools via self-service interfaces.

- Service-oriented architectures (SOA). They define applications whose components are implemented as independent services. SOA can be flexibly tied together and orchestrated, and they can communicate asynchronously via messaging systems in a loosely coupled configuration. With cloud computing, virtualized IT infrastructures, platforms, and entire applications are implemented as services and made available for consumption in SOA. Consumers or clients for these services can be other services, applications or final customers. Interoperability is an essential feature of SOA, so heterogeneus service consumers and service providers must be able to interact across different platforms or programming languages.

- Web services. In cloud computing, independent services aggregated in SOA can be orchestrated not only over local networks, but also over the internet, allowing applications to be built on the basis of globally distributed resources. This situation results in enhanced availabilty because servers and data stores can be replicated on several locations in order to keep them close to many different customers. It also increases performance and scalability because data management and processing tasks can be delegated, when required, on multiple servers from one or more cloud providers. However, communications over the internet are likely to experience slow response times and unreliable connections. A practical solution to this problem is to implement the SOA application or component as a web service, able to be located on the internet with a Uniform Resource Identifier (URI), and to asynchronously communicate using standardized requests over internet-based protocols. Two common approaches for this communication are transferring XML-compliant data packages according to messaging standards like Simple Object Access Protocol (SOAP), or Representational State Transfer (REST).

Organizational Prerequisites

In addition to the aforementioned technology-based fundamentals, a successful implementation of cloud computing requires a particular organizational context centered on the notion of *service*. This special context can be illustrated with native cloud service companies like Uber, Amazon, Netflix, or Airbnb, which operate cloud applications able to effectively manage widely distributed physical resources, like cars, data centers, audiovisual contents, or accomodations, for serving customers that value more making use of a product than owning it. Three key aspects can be found on the root of this type of operation: a service-dominant logic, a networked business model, and the concept of service ecosystem. These features can be summarized from [Castro-Leon and Harmon, 2016] as follows:

- Service-dominant logic. It is a particular method of thinking business processes where the primary key is not a concrete good or product, but the intangible user experience the good or product conveys. For cloud computing to prosper the traditional good-dominant logic, that views services as non-core add-ons to the value of products, have to be replaced in favor of a service-dominant logic, where service is considered as an essential value-creating process involving knowledge and skills to create primary benefits for another party.

- Networked business model. It consists on leveraging multi-sided service platforms, such as cloud-based software, social networks, search engines, and mobile operating systems, in order to allow engagement beyond the usual provider-customer dyad, in the form of direct or indirect interactions between suppliers, partners, developers, advertising agencies, employees, distributors, agents, competitors, shareholders, etc., sometimes simultaneously. As a result, business contexts and processes, typically fixed and locally enclosed, become fluid and distributed over global landscapes.

- Service ecosystem. It comprises the sum of all of the actors in a networked business as well as the service exchanges that those actors perform in a collaborative way to create value for mutual benefit. Actors that populate the service ecosystem can be individuals, online communities, organizations, and even machines, and they primarily integrate resources like knowledge, skills, capabilities, and technologies, according to shared institutional arrangements.

Environment Summary

The environment where ITSFCC will reside is shaped by the intention of exploiting the five *essential characteristics* of cloud computing to enhance the intelligent time series forecasting process. In order to achieve this objective a cloud deployment model has to be selected and prepared as the ground for the development and evaluation of multiple prototypes. The most straightforward option for experimenting with cloud computing is indeed the *public cloud* deployment model, because cloud providers have already implemented the resources needed to develop, operate, and manage robust applications, making such resources available at a fraction of the cost and effort required to provision them in a private or community cloud. Regarding the service model, ITSFCC will render two different modes based on its development stage and the corresponding user type. At a development stage, when used only by the academic stakeholders affiliated to the Work Group, ITSFCC will conform to a *Platform-as-a-Service* model, suitable for the deployment of consumer-created, intelligent time series forecasting applications. Once at production stage, and made available to non-specialized customers coming from a wider service ecosystem, it will take the form of a *Software-as-a-Service* tool, more specifically, a Forecasting-as-a-Service (FaaS) one. Next section deals with the identification of the system concerns associated to these characteristics.

4.2 Identification of System Concerns

The identification of the concerns placed to ITSFCC by its stakeholders follows the conventions described in Subsection 3.2.1. The initial collection of values for ITSFCC's quality attributes presented in Subsection 3.2.1 remains valid for the application components listed here, and only aggregates considerations for additional users able to analyze forecasting results via web services. Consequently, the following quality attribute values are added to the original collection:

- Scalability: work load of 20 to a peak of 50 users, located anywhere in Mexico, either querying or visualizing completed forecasting results.
- Availability: ITSFCC will have to be available for 99% of total requests for analyzing forecasting results.
- Performance: once completed, forecasting results must be accessible, via a user interface, in less than one second.

4.2.1 Rapid Elasticity

Processing components of ITSFCC are likely to deal with huge numerical tasks related to time series analysis and production of complex forecast models in a Big Data environment. In addition, the need for running multiple experiments over the data for model evaluation or enhancement will constantly modify processing workloads. Elasticity, as the ability to easily provision or decommission resources in order to meet computing demand, is a fundamental feature of ITSFCC. Moreover, scaling resources outward or inward must be done quickly so as to avoid performance issues when workload increases or incurring in superfluous costs when it decreases.

- Distributed Application: Essential application components of ITSFCC must be able to scale outward or inward[7] independently. Processing components of ITSFCC place the higher requirement for scalability due to intensive variable workloads associated with intelligent time series forecasting. Primary stakeholders of ITSFCC are the academic

[7]An application or component is scaled outward by increasing the number of virtual servers where it is executed, in order to deal with a higher workload. If the workload decreases, the application or component is scaled inward by reducing the number of virtual servers.

research group in the Work Group, so user interface and data management components are currently far less demanding for scalability. However, present conditions might change in the future if the system is made available to new users as part of a wider service ecosystem. Performance: Sub-one hour intervals for task completion. Availability: 0.99 during on-request operation. Scalability: Peak load of 20 concurrent forecast processes.

- Multi-variable Elasticity Manager: Load measurements in terms of CPU utilization, quantity of synchronous accesses to the user and application interface components, and quantity of asynchronous accesses to the processing components must be continuously monitored in real time, and stored for further in-depth analytics. Real time monitoring provides data triggers for scaling decisions while in-depth analysis supports overall performance evaluation of the system. ITSFCC components must be able to automatically scale outward or inward in order to meet workload demand, using results obtained from workload monitoring to determine the number of virtual servers required for deploying interface, processing, and data components. Additionally, a set of rules for provisioning/decommissioning must be implemented, based on the behavior of the monitored variables over standardized periods of time. This is because, for instance, a new server should not be immediately added when CPU utilization raises to a pre-defined threshold, but when the expected behavior of this variable suggests the threshold will be steadily reached if no server is added. Availability: As previously stated. Supportability: Application component must be ready from shutdown to fully functional state in sub-ten minute periods.

4.2.2 Secure Operation

In cloud computing, a special concern for security arises from the fact that customers or tenants have no control or visibility on the portions of the physical infrastructure their platforms and applications run on. [Yeluri et al., 2012] explore challenges in deploying and managing services in a cloud infrastructure from a security perspective and identify the following five key drivers for cloud security: (1) identity management, (2) visibility, compliance, and monitoring, (3) data discovery and protection, (4) architecture, and (5) automation policy and orchestration. The specific context of ITSFCC places a security condition that can be primarily handled with a narrower approach, which considers only the first, third, and fourth of these key drivers.

- User Identity Manager: ITSFCC must be able to verify the identity of users and other applications with which it interacts, in terms of a valid set of access rights granted to each one of them over the different resources of the system. Availability and Supportability: As previously stated. Security: All data must be transferred over secure connections. All data must be stored behind a firewall accessible only from the data access or the processing components.

- Data Encryption: ITSFCC must be able to apply different encryption levels to digital data when *at rest* (in storage systems and databases), *in transit* (flowing over a network), and *in use* (in CPU and memory). In addition, the system must be able to protect data in transit using encrypted connections like HTTPS, FTPS, TLS, or SSL, in order to deal with its higher vulnerability. Availability and Security: As previously stated.

- Firewall Protection: ITSFCC application components must be implemented over virtual local area networks (VLANs) to define subnets or network segments on which virtual servers may communicate directly with each other. Then, it must be possible to place virtual firewalls between these subnets, and to configure them with specific routing tables and access rules in order to securely expose interfaces for users or applications in public subnets, while keeping processing and data components hidden in private subnets. Availability, Security and Supportability: As previously stated.

4.2.3 Forecasting-as-a-Service Operation

As a novel and comprehensive research project, ITSFCC condenses valuable knowledge and skills from an important group of academics affiliated to the Work Group. Therefore, the system comprises significant data science capabilities that can be made available to a wider service ecosystem in the form of a Software-as-a-Service tool, more specifically a Forecasting-as-a-Service (FaaS) one. Multiple entities having interests in intelligent time series forecasting like energy providers, universities, public service institutions, and entrepreneurs can collaboratively integrate their specific resources into ITSFCC operation in order to create value for a mutual benefit. Taking ITSFCC to an adequate FaaS operation primarily involves the application components listed below.

- RESTful Interfaces: ITSFCC components, especially those related to the user and application interfaces, must be exposed as RESTful web services, which means they must operate in response to HTTP request messages composed by a HTTP method (i.e. POST, GET, PUT, DELETE), the URI that locates the service over the network, and the request body and header, which describe the data required for the operation. Usually authentication and metadata are provided in the request header while operation details are provided in the request body [Vyas, 2015]. The main advantage of this approach is that the data handled by the application component, commonly referred as its *state*, is provided in every request message, so it does not have to be kept in the server side. The application component becomes *stateless* and, consequently, multiple request messages coming from one or more clients can be handled by arbitrary servers hosting the application component, which is now easily scalable and replaceable in case of failure, on a cloud environment. Availability: 0.99 during continuous operation. Scalability: Peak load of 50 concurrent users. Security and Supportability: As previously stated.

- Pipes-and-Filters Processing: Academic stakeholders of ITSFCC will be able to operate the system by manually executing each one of its application components, in order to get intermediate and final results, as well as to run advisable variations from the main sequence of operations, following a PaaS service model. When made available as a FaaS tool, ITSFCC must be able to automatically execute the complete sequence of processing operations, based on requests from customers or tenants who are likely to expect minimal or null interaction with the application components. A *pipes-and-filters* processing model is recommended for data-centric processing of an application and consists of filters which provide a certain function that is performed on input data and produces output data after processing. Multiple filters are interconnected with pipes ensuring that the output of one filter is fed to the next filter in a processing chain. In a distributed application, such as ITSFCC, filters map to application components that provide a certain function and are interconnected using communication services provided by a cloud [Fehling et al., 2014]. Availability: 0.99 during continuous operation. Scalability: Peak load of 20 concurrent forecast processes. Security and Supportability: As previously stated.

- Engagement Layer: Academic stakeholders of ITSFCC are able to get insight into the results from the system with a rapid glance at numerical outcomes or a few graphic representations. However, once available as a FaaS tool ITSFCC must provide tenants with a *visualization layer* built on top of the system to get such insight. This application component, which [Nandi, 2015] identifies as *engagement layer* when immersed in a data-intensive application, must give tenants access to a comprehensive set of graph styles, as well as to customized dashboards for at-a-glance views of forecasting key performance indicators. The engagement layer must be exposed as a RESTful web service so as to be accessed via modern web browsers. It must also efficiently render visual interactive analytics over huge data sets. Performance: Sub-one second response time for graphics or dashboards updates. Availability: 0.99 during continuous operation. Peak load of 50 concurrent users. Security and Supportability: As previously stated.

4.2.4 Summary of System Concerns

Fig 4.2 represents a summary of the system concerns identified in this section. It shows broad features required by ITSFCC to properly leverage cloud computing potentials, the application components required to achieve such features, and the corresponding software quality attributes. The figure clearly shows more quality attributes are called for once ITSFCC is available as a FaaS tool. Next section presents the software patterns suggested to deal with the identified system concerns for cloud computing operation.

4.3 Software Patterns Related to System Concerns

This section presents the software patterns suggested to address the system concerns identified in the previous section; it follows the classification established in [Fehling et al., 2014], so, besides proper architectural patterns, cloud

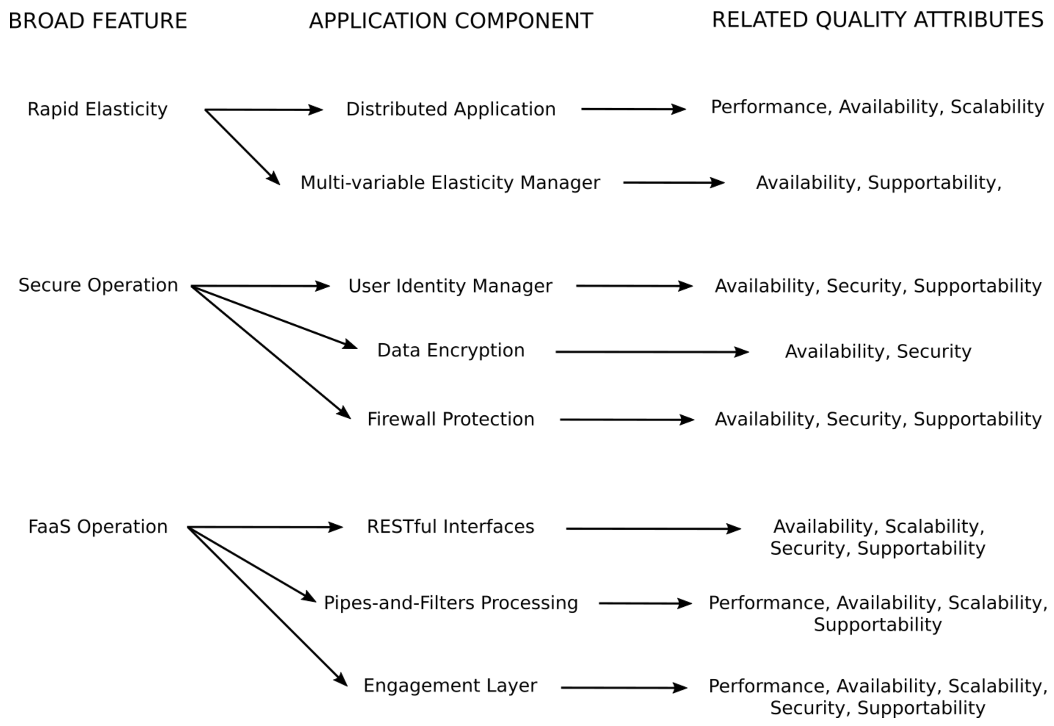
BROAD FEATURE
APPLICATION COMPONENT
RELATED QUALITY ATTRIBUTES
Rapid Elasticity
Distributed Application
Performance, Availability, Scalability
Multi-variable Elasticity Manager
Availability, Supportability,
Secure Operation
User Identity Manager
Availability, Security, Supportability
Data Encryption
Availability, Security
Firewall Protection
Availability, Security, Supportability
FaaS Operation
RESTful Interfaces
Availability, Scalability, Security, Supportability
Pipes-and-Filters Processing
Performance, Availability, Scalability, Supportability
Engagement Layer
Performance, Availability, Scalability, Security, Supportability

Figure 4.2: Summary of system concerns related to Cloud Computing

offering patterns are also listed. As in Chapter 3, subsection headers identify the categories used to aggregate individual patterns, which are described in terms of their intent and ITSFCC context attributes. All patterns in this section come from the aforementioned source.

4.3.1 Cloud Offering Patterns

Patterns in this subsection correspond to virtual resources that are offered via a self-service interface over a network.

Elastic Infrastructure

Intent: Hosting of virtual servers, disk storage, and configuration of network connectivity is offered via a self-service interface over a network.

ITSFCC Context: This pattern is intended to provide elasticity to the servers, storage systems, and networking elements in the Distributed Application.

Elastic Platform

Intent: Middleware for the execution of custom applications, their communication, and data storage is offered via a self-service interface over a network.

ITSFCC Context: This pattern is intended to provide elasticity to the databases in the Distributed Application, as well as to customized programs for visualization in the Engagement Layer.

Relational Database

Intent: Data is structured according to a schema that is enforced during data manipulation and enables expressive queries of handled data.

ITSFCC Context: This pattern is intended to provide efficient data management for the User Identity Manager, including user specification, permissions granted over different resources, security groups and roles, etc. It also provides a way to store metadata from the request messages reaching the RESTful Interfaces.

Virtual Networking

Intent: Networking resources are virtualized to empower customers to configure networks, firewalls, and remote access using a self-service interface.

ITSFCC Context: This pattern is intended to provide the virtual LANs and other networking resources required to implement the Firewall Protection of ITSFCC.

Message-Oriented Middleware

Intent: Asynchronous message-based communication is provided while hiding complexity resulting from addressing, routing, or data formats from communication partners to make interaction robust and flexible.

ITSFCC Context: This pattern is intended to provide the message queues that act as *pipes* in the Pipes-and-Filters Processing model. Under this approach, the application components act as the independently operating *filters*.

4.3.2 Cloud Application Architecture Patterns

Patterns in this subsection correspond to architectural patterns that describe how applications have to be designed to benefit from a cloud environment.

Data Access Component

Intent: Functionality to store and access data elements is provided by special components that isolate complexity of data access, enable additional data consistency, and ensure adjustability of handled data elements to meet different customer requirements.

ITSFCC Context: This pattern is intended to facilitate access operations for Data Encryption, as well as for handling application state for the application components exposed as RESTful Interfaces.

Stateless Component

Intent: State is handled external of application components to ease their scaling-out and to make the application more tolerant to component failures.

ITSFCC Context: This pattern is intended to provide a model for the implementation of ITSFCC application components, so they can be exposed as RESTful Interfaces.

Loose Coupling

Intent: A communication intermediary separates application functionality from concerns of communication partners regarding their location, implementation platform, the time of communication, and the used data format.

ITSFCC Context: This pattern is intended to provide an operational basis for the ability of ITSFCC to scale its components independently, as required by the Distributed Application.

User Interface Component

Intent: Interactive synchronous access to applications is provided to humans, while application-internal interaction is realized asynchronously when possible to ensure loose coupling. Furthermore, the user interface should be customized to be used by different customers.

ITSFCC Context: This pattern is intended to provide a basis for the interactive operation of the Engagement Layer.

4.3.3 Cloud Application Management Patterns

Patterns in this subsection correspond to architectural components that enable the automated execution of management processes handling application components and system resources.

Elasticity Manager

Intent: The utilization of IT resources on which an elastically scaled-out application is hosted, for example virtual servers, is used to determine the number of required application component instances.

ITSFCC Context: This pattern is intended to provide a measurement of the system workload in terms of the CPU utilization for the Multi-variable Elasticity Manager component.

Elastic Load Balancer

Intent: The number of synchronous accesses to an elastically scaled-out application is used to determine the number of required application component instances.

ITSFCC Context: This pattern is intended to provide a measurement of the system workload in terms of the synchronous accesses to the user and application interface components for the Multi-variable Elasticity Manager.

Elastic Queue

Intent: The number of asynchronous accesses via messages to an elastically scaled-out application is used to adjust the number of required application component instances.

ITSFCC Context: This pattern is intended to provide a measurement of the system workload in terms of the asynchronous accesses to the processing components for the Multi-variable Elasticity Manager.

Elasticity Management Process

Intent: Application component instances are added automatically to an application to cope with increasing workload. If the workload decreases application component instances are removed respectly.

ITSFCC Context: This pattern is intended to provide a basis for the detailed implementation of the rules for provisioning/decommissioning application component instances in the Multi-variable Elasticity Manager. [Fehling et al., 2014] illustrate the use of the Business Process Model and Notation (BPMN) language for expressing this pattern.

Summary

This chapter presented the concept of *cloud computing* as the second part, along with *intelligent series forecasting*, of the foundation for ITSFCC environment. It also presented the system concerns as a set of broad features ITSFCC must include to properly leverage cloud computing potentials, a set of application components that support the features, and a set of precise software quality attributes linked to the application components. Finally, cloud computing patterns were used to describe architectural elements suggested to address the identified concerns. Next chapter will integrate the architectural elements described in Chapters 3 and 4 into the definitive architecture characterization for ITSFCC, in accordance with the specifications developed in Chapter 2.

Chapter 5

ITSFCC Architecture

This chapter presents the detailed architecture characterization of ITSFCC, in compliance with the specifications covered in Chapter 2. Section 5.1 presents a set of text-based documents containing the architecture identification and overview, the identification of stakeholders, and the identification of concerns. Section 5.2 presents the architecture model, which conforms to a process-based architecture view and is governed by a block diagram model kind. The overall architecture of ITSFCC, as well as each one of its application components are depicted in a comprehensive set of graphics, which are described in detail by explanatory texts.

5.1 Basic Documentation

This section covers the basic documentation for ITSFCC's architecture characterization, in compliance with the specifications listed in Chapter 2. It comprises the following text-based documents:

- Architecture identification and overview.
- Identification of stakeholders.
- Identification of concerns.

5.1.1 Identification and Overview

- Date of issue: March, 2018.

- Status: Model completed and ready for prototyping in a public cloud.
- Authors: José Luis García Nava and Juan José Flores Romero.
- Issuing organization: Data Science Research Work Group at the Graduate School of Electrical Engineering, Universidad Michoacana de San Nicolás de Hidalgo (The Work Group).
- Change history: Null (no changes have been made to the architecture characterization).
- Context: The implementation of a set of Artificial Intelligence and Machine Learning-based methods for time series forecasting available to multiple tenants in the form of a Software-as-a-Service tool.
- Glossary: All of the following required concepts can be reviewed in chapters 2, 3, and 4: software architecture, architecture description, architecture characterization, system, environment, stakeholder, concern, software quality attributes, performance, availability, scalability, security, supportability, architecture view, architecture model, model kind, software pattern, intelligent time series forecasting, hybrid-operation forecasting, context-aware forecasting, Nearest Neighbors Differential Evolution, cloud computing, virtualization, service-oriented architecture, web services, service-dominant logic, networked business model, service ecosystem, Infrastructure-as-a-Service, Platform-as-a-Service, Software-as-a-Service, Forecasting-as-a-Service.
- References: The references this architecture characterization conform to the References section of this thesis.

5.1.2 Identification of Stakeholders

Researchers and graduate students affiliated to the Work Group are the primary stakeholders of ITSFCC, as well as the only users, operators, developers, builders, and maintainers of the system during its development stage. Once ITSFCC become available as a FaaS tool, new stakeholders will be incorporated to conform a wider service ecosystem. It is expected that energy providers, universities, public service institutions, and entrepreneurs will participate in the future as users, acquirers, and suppliers of the system.

5.1.3 Identification of Concerns

So far, ITSFCC concerns have been expressed as a set of broad features the system must incorporate in order to deal with the problems related to intelligent time series forecasting (Chapter 3), and to properly leverage the potentials of cloud computing (Chapter 4). Application components required to address the broad features were listed, and then mapped to specific software quality attributes as summarized in Fig. 3.3 and in Fig. 4.2. The software quality attributes defined for ITSFCC's application components can be aggregated as follows:

1. Performance: Sub-one hour intervals for processing tasks completion.
2. Performance: Sub-one second response time for graphics and dashboards update on the *Engagement Layer* component.
3. Scalability: Peak load of 20 concurrent forecast processes on processing components.
4. Scalability: Peak load of 50 concurrent sessions of the user interface components.
5. Availability: 99% during on-request operation of the data repositories and the processing components.
6. Availability: 99% during continuous operation of the user interface components.
7. Security: All extraction operations on the Multi-source Data Extraction component must be performed over secure connections.
8. Security: All data must be transferred over secure connections.
9. Security: All data must be encrypted when *at rest*[1] or *in transit*[2].
10. Supportability: All application components must be ready from shutdown to fully functional state in sub-ten minute periods. This is particularly crucial for the *Publication-Subscription Channel* and the *Multivariable Elasticity Manager*, which are the core management components of ITSFCC.

[1]Data at rest refers to inactive data that is stored in any digital form.
[2]Data in transit refers to data that flows over a network

5.2 Architecture Model

This section presents the architecture model for ITSFCC, which exhibits the following characteristics:

- It conforms to an architecture view based on the concept of *process*, defined as a group of tasks that form an executable unit which can be (a) tactically controlled (started, recovered, reconfigured, shut down, and so on), (b) replicated to distribute processing load or improve system availability, and (c) partitioned into a set of independent tasks.

- It is governed by the *block diagram* model kind, which allows to represent the system's architecture as a high-level structure composed of interconnected blocks. In this modelling technique blocks identify processes while connections identify messages that the processes exchange.

- As a consequence of the previous items, this architecture model contributes to efficiently address concerns related to *non-functional requirements* of the system. As long as no business or technical constraints are considered in the scope of this work, non-functional requirements of the system are equivalent to its software quality attribute values.

The ITSFCC architecture model is depicted in Fig. 5.1. It includes the following ITSFCC application components:

1. Multi-source Data Extraction.

2. Data Encryption.

3. User Identity Manager.

4. Engagement Layer.

5. Publication-Subscription Channel.

6. Multi-variable Elasticity Manager.

7. Graph Representation.

8. RESTful Interfaces.

9. Time Series Repository.

10. Forecast Model Repository.

11. Batch Processing.

The individual diagrams of the above application components are presented in subsequent pages. A total of four ITSFCC application components are not included in Fig. 5.1 because of the following reasons:

1. Distributed Application. This functionality does not depend on a single application component but on the ability of all ITSFCC's components to scale independently.

2. Parallel Implementation. This application component is intended to operate in a side-by-side way with the Batch Processing component, and to eventually take over the serial processing-based functionalities of Batch Processing into a parallel operation. Therefore it was considered redundant to place this component on the general architecture diagram.

3. Pipes-and-Filters Processing. This functionality does not depend on a single application component, but on the integration of five of them into an automated process that is executed on schedule or in response to a specific request. A separate diagram for this functionality is provided.

4. Firewall Protection. This functionality does not depend on a single application component, but on the implementation and regulation of a Virtual Local Area Network that includes all of the ITSFCC's components. A separate diagram for this functionality is provided.

Based on the ITSFCC architecture model, a graphic concern map was prepared to clarify how the requirements on software quality attributes impact the system. Depicted in Fig. 5.2, the ITSFCC concern map shows a simplified block representation of the architecture, on top of which six categories that logically group application components are represented by enclosed areas, as follows:

1. Data Ingestion: includes Multi-source Data Extraction component.

2. Security: includes Data Encryption and User Identity Manager components.

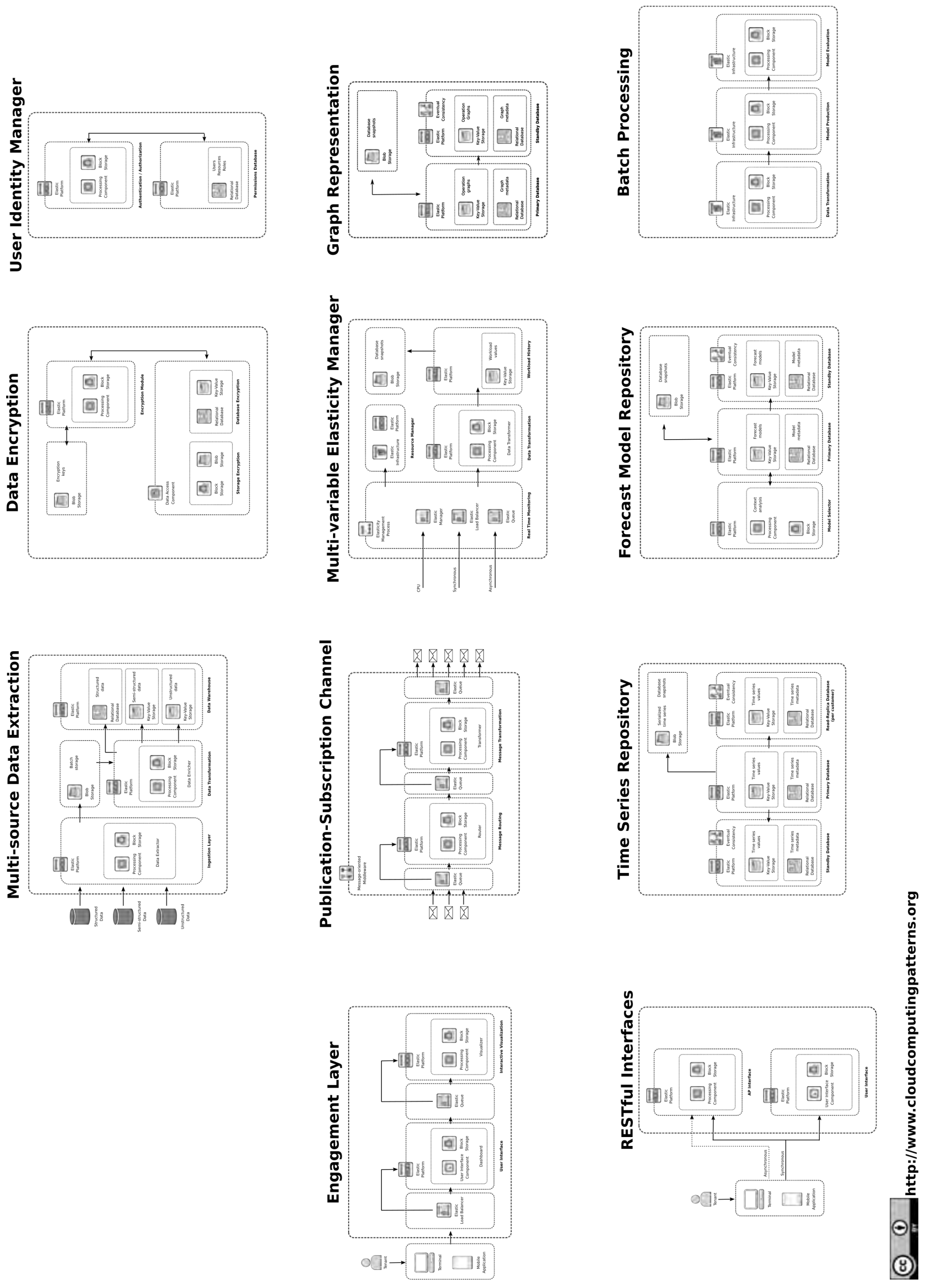
Multi-source Data Extraction
Data Encryption
User Identity Manager
Publication-Subscription Channel
Multi-variable Elasticity Manager
Graph Representation
Engagement Layer
Time Series Repository
Forecast Model Repository
Batch Processing
RESTful Interfaces
http://www.cloudcomputingpatterns.org

Figure 5.1: ITSFCC Architecture

3. Interfaces: includes RESTful Interfaces and Engagement Layer components.

4. Management: includes Publication-Subscription Channel and Multi-variable Elasticity Manager components.

5. Data Repositories: includes Time Series Repository and Forecast Model Repository components.

6. Processing: includes Graph Representation, Batch Processing, and Parallel Implementation components.

The concern map of ITSFCC was completed by labeling each quality attribute value inside the component category it primarily impacts. Important information can be easily derived from the examination of this concern map, for instance that performance and scalability requirements placed to application components in the *Interfaces* category are higher than those placed to components in the *Processing* category. Likewise, security is mainly enforced in the *Data Ingestion* and *Security* categories, which basically encompass the data management operations. This concern map is intended to achieve higher detail with further refinements on the mapping from application components to software quality attributes to be produced by the continuous evaluation of the ITSFCC architecture.

The following subsections describe the application components of ITSFCC as elements of the aforementioned architecture model: the subsection header identifies the application component, then the subsection body describes the software patterns the component includes, the tasks it performs, and the relationships it establishes to other application components.

5.2.1 Distributed Application

Fig. 5.1 shows ITSFCC's application components implemented as loosely coupled services communicated by a central Publication-Subscription Channel. As a result of this decoupled architecture all of the components can be scaled outward or inward independently via a central Multi-variable Elasticity Manager.

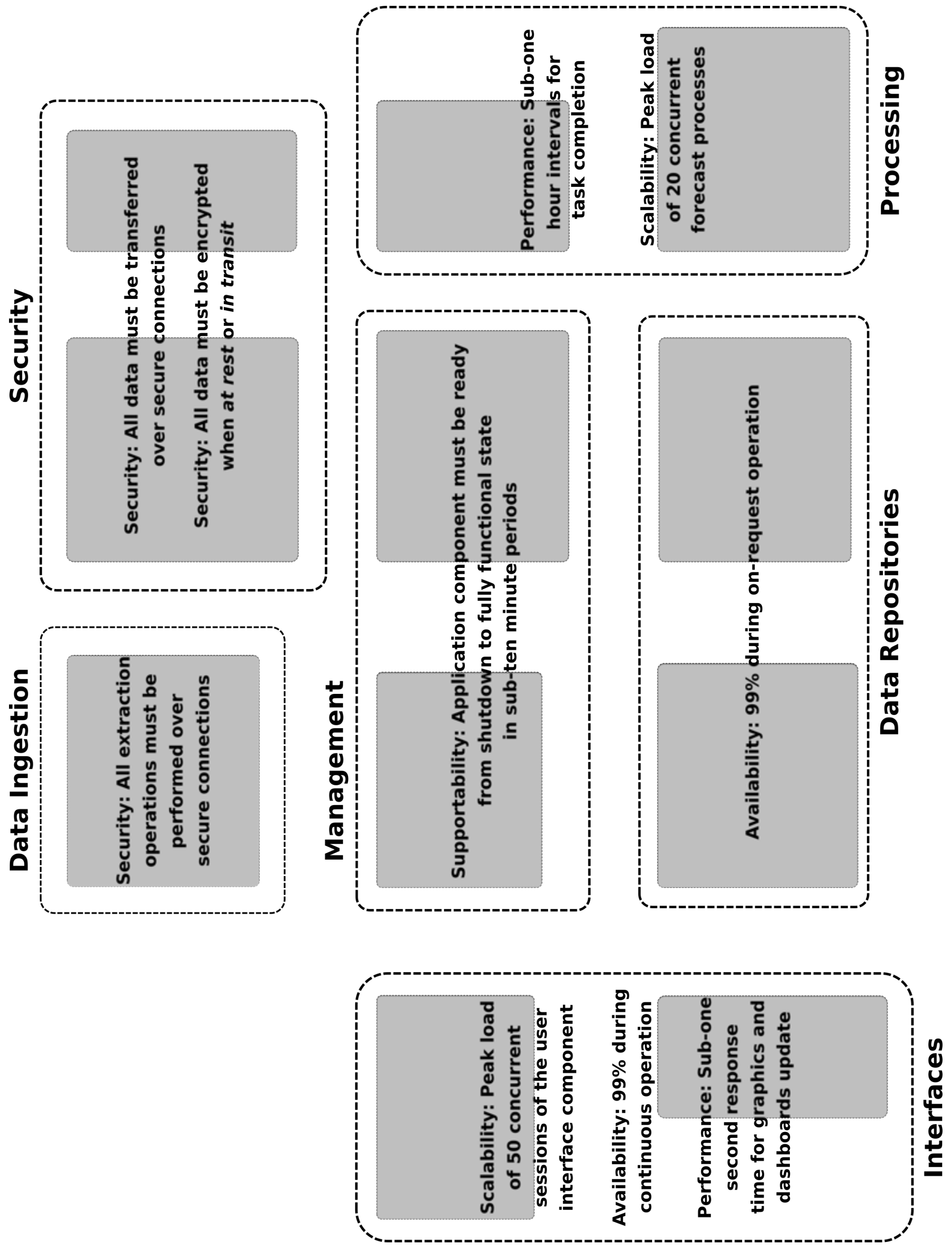
Data Ingestion
Security: All extraction operations must be performed over secure connections
Security
Security: All data must be transferred over secure connections
Security: All data must be encrypted when *at rest* or *in transit*
Management
Supportability: Application component must be ready from shutdown to fully functional state in sub-ten minute periods
Processing
Performance: Sub-one hour intervals for task completion
Scalability: Peak load of 20 concurrent forecast processes
Data Repositories
Availability: 99% during on-request operation
Interfaces
Scalability: Peak load of 50 concurrent sessions of the user interface component
Availability: 99% during continuous operation
Performance: Sub-one second response time for graphics and dashboards update

Figure 5.2: ITSFCC's concerns mapped to application component categories

5.2.2 Multi-source Data Extraction

The Multi-source Data Extraction component, depicted in Fig. 5.3, implements a *data lake* pattern for gathering time series and their specific contexts. Structured data (relational databases and CSV files), semi-structured data (XML and JSON files), and unstructured data (e-mail messages, word processing documents, photos, etc.) that describe variables related to the forecasting process are extracted from external sources and ingested into a *blob storage*. Data is then transformed to comply with ITSFCC specifications and stored in a data warehouse conformed with elastic *relational databases* (SQL) and *key-value storage* (NoSQL). Elastic *processing components* configured on *block storage* (operating system, software, fast-access data storage) are used for implementing data extraction and transformation. Due to the huge, constantly increasing size of the data sets, databases are not replicated, leaving *blob storage* as the only backup for extracted information.

5.2.3 Data Encryption

The Data Encryption component, depicted in Fig. 5.4, implements a *data access component* pattern to isolate complexity of data access over *blob storage*, *block storage*, *relational database*, and *key-value storage* instances distributed over the complete ITSFCC architecture. Data is encrypted when saved to storage elements and decrypted when retrieved so as to keep it safe at rest. Encryption is implemented in an elastic *processing component* configured on *block storage* (operating system, software, fast-access data storage) following the Advanced Encryption Standard (AES) algorithm in Galois/Counter Mode (GCM), known as AES-GCM. This is a symmetric-key algorithm that can be used with 128, 192, and 256-bit secret keys [Dworkin, 2007]. Encryption keys and configuration files are kept in *blob storage*.

5.2.4 User Identity Manager

The User Identity Manager component, depicted in Fig. 5.5, implements an elastic *relational database* to store information about users, resources, and the permissions that link them. Security roles and policies can also be kept in the database. Authentication and authorization processes are implemented in an elastic *processing component* configured on *block storage* (operating system, software, fast-access data storage).

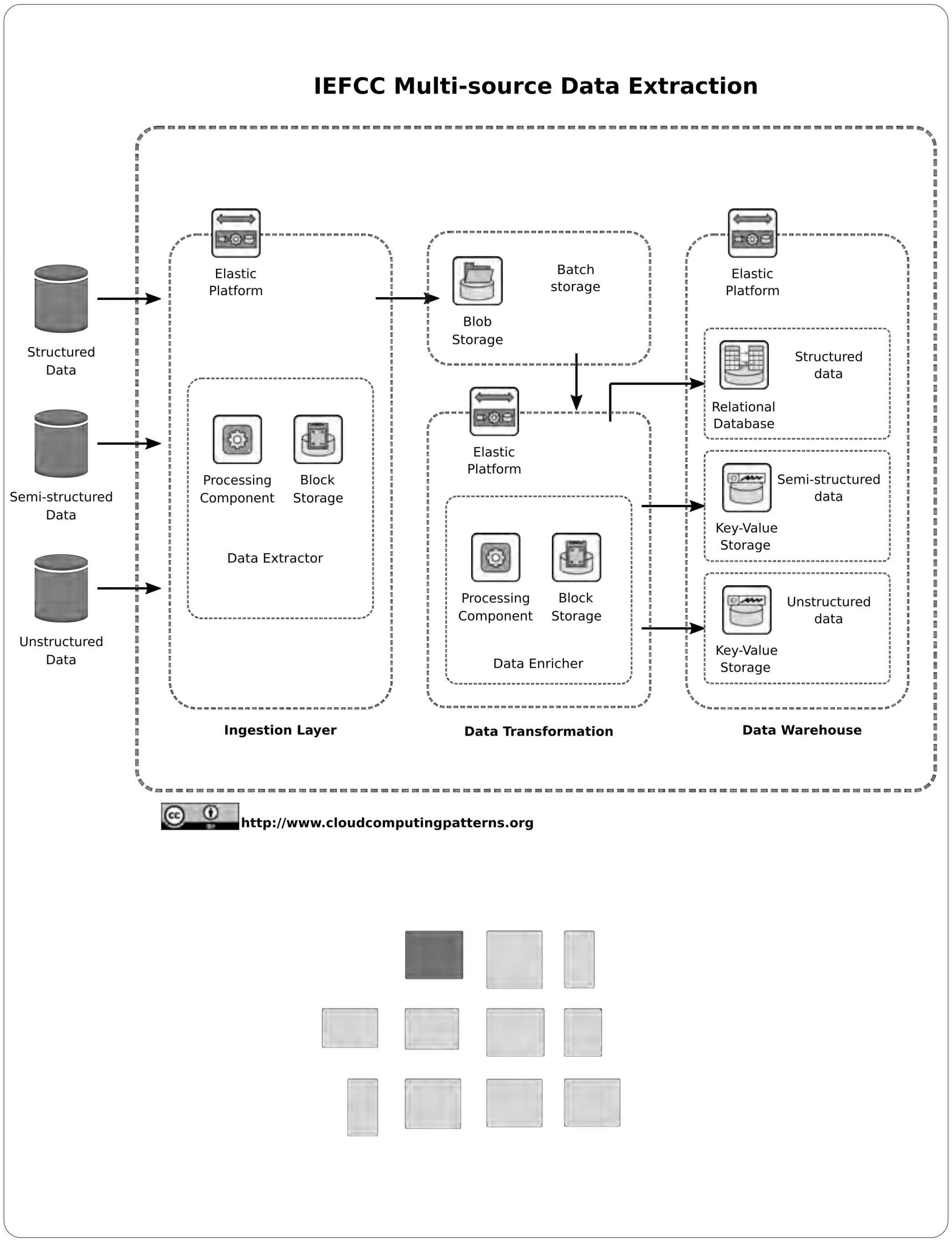

Figure 5.3: Multi-source Data Extraction

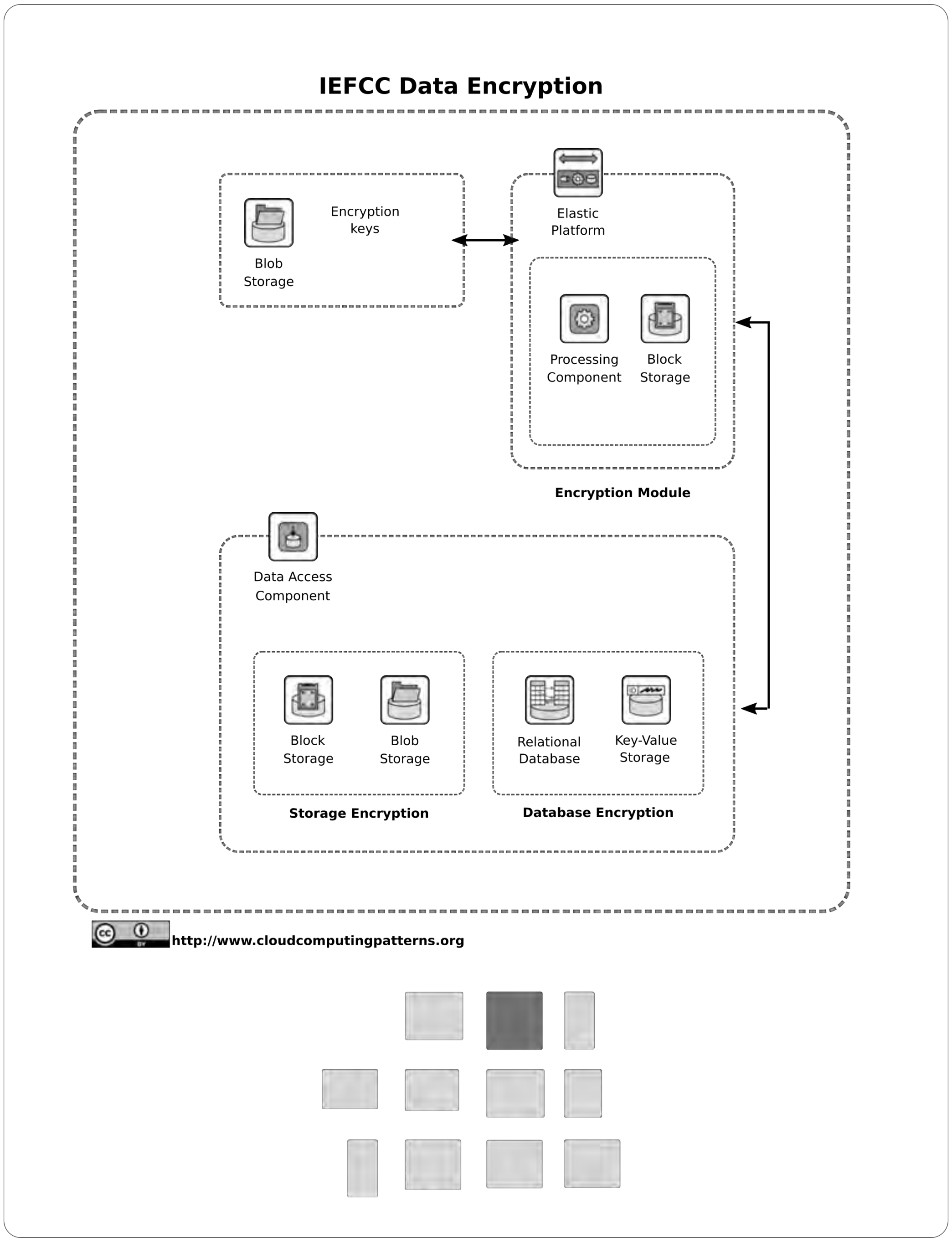

Figure 5.4: Data Encryption

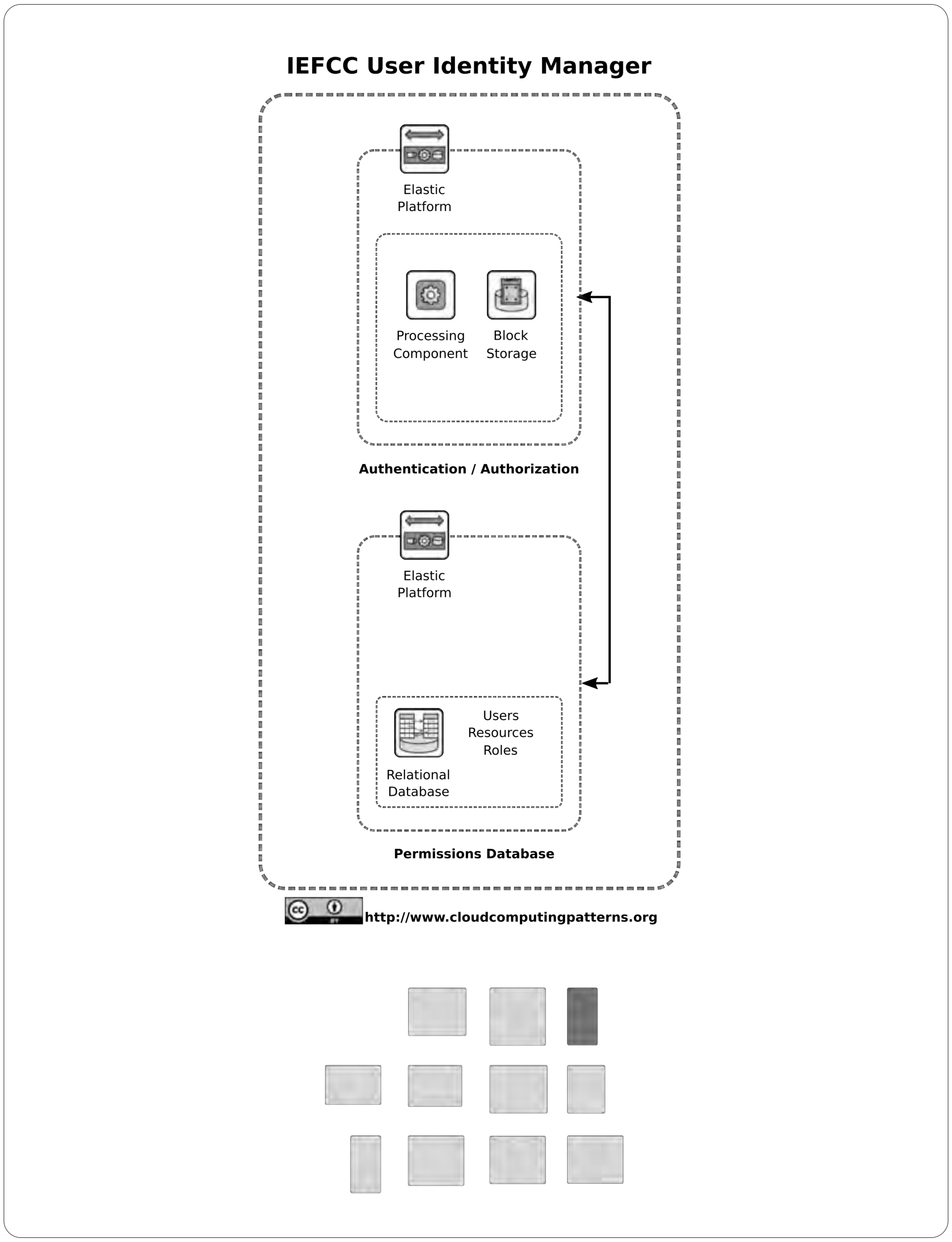

Figure 5.5: User Identity Manager

5.2.5 Engagement Layer

The Engagement Layer component, depicted in Fig. 5.6, is intended to provide interactive graphics and dashboards via modern web browsers or mobile applications. It implements an elastic *user interface component* managed with an *elastic load balancer* for dashboard operation, as well as an elastic *processing component* managed with an *elastic queue* for interactive visualization. Both components are configured on *block storage* (operating system, software, fast-access data storage), and follow a *stateless component* pattern for ensuring *loose coupling*. Consequently, the user interface component must keep its session state on the client side and send it to the processing server with each request.

5.2.6 Publication-Subscription Channel

The Publication-Subscription Channel component, depicted in Fig. 5.7, asynchronously communicates all of the ITSFCC components by exchanging messages. It implements a *message-oriented middleware* pattern where incoming messages are passed to an *elastic queue*. The number of messages arriving to the queue is used to scale an elastic *processing component* which routes messages to intended receivers. A second *elastic queue* is used to scale another elastic *processing component* which handles message format transformation to comply with each receiver's specifications. Both processing components are configured on *block storage* (operating system, software, fast-access data storage). Once routed and transformed, messages are put in an *elastic queue* where receivers pick them up.

5.2.7 Multi-variable Elasticity Manager

The Multi-variable Elasticity Manager component, depicted in Fig. 5.8, implements an *elasticity management process* pattern to scale the *elastic platform* and *elastic infrastructure* instances distributed over the complete ITSFCC architecture, based on real time monitoring of three variables: CPU utilization of virtual servers is managed by an *elastic manager*, the number of synchronous accesses to interface components is managed by an *elastic load balancer*, and the number of asynchronous accesses to processing components is managed by an *elastic queue*. In addition, all this data is transformed to key-value pairs by an elastic *processing component* configured on *block stor-*

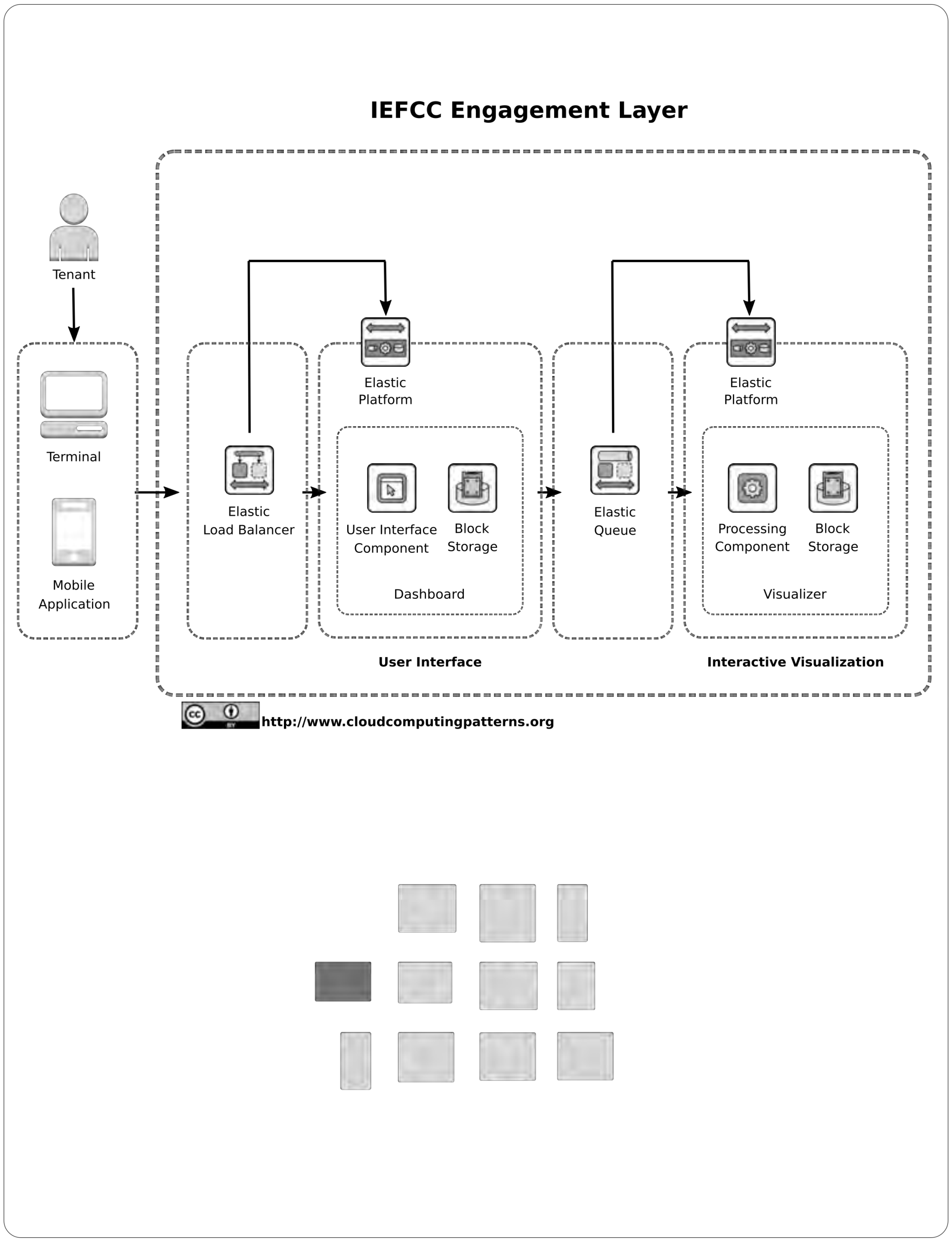

Figure 5.6: Engagement Layer

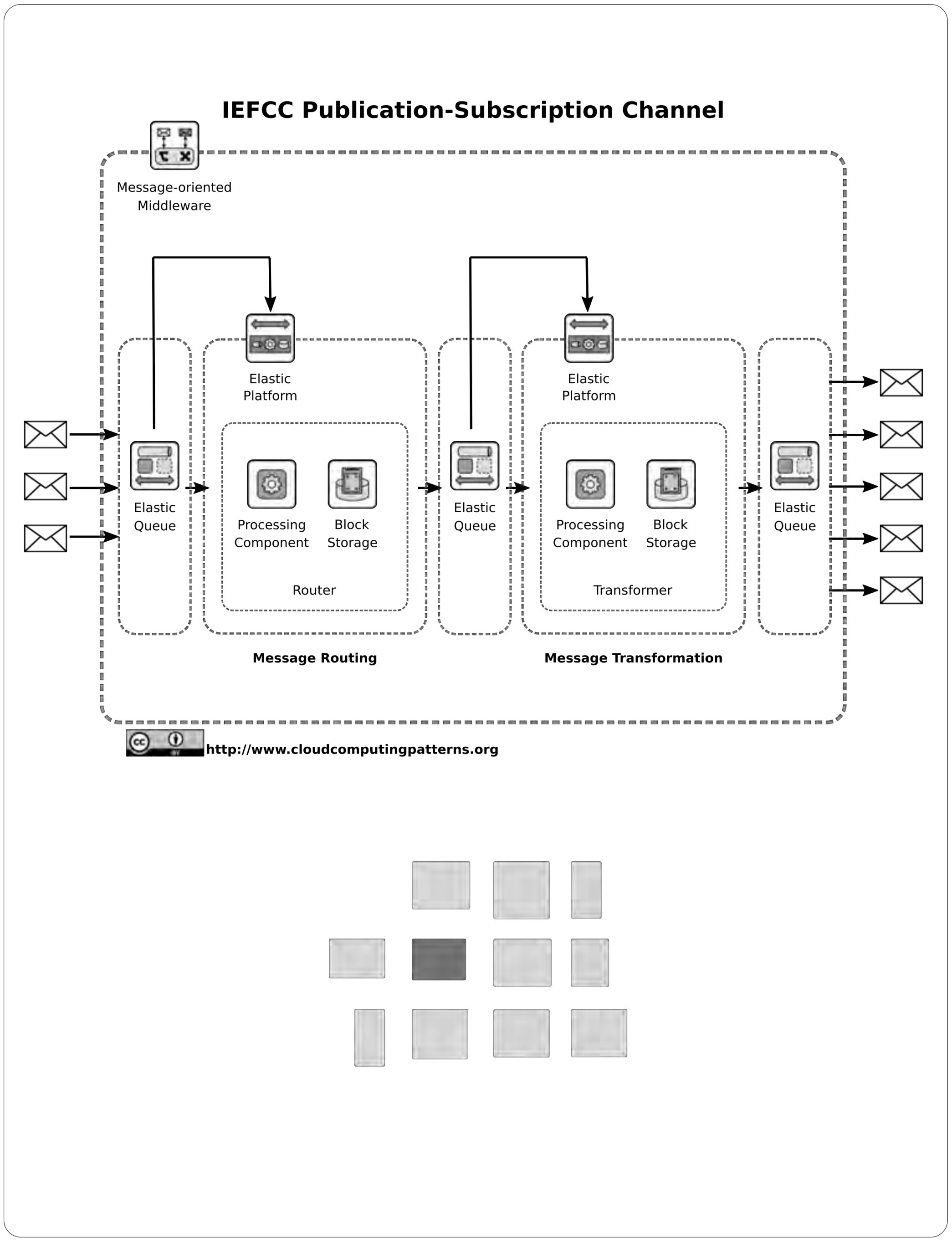

Figure 5.7: Publication-Subscription Channel

age (operating system, software, fast-access data storage), then stored as workload history values in an elastic *key-value storage* for further analytics. *Blob storage* is used to keep workload history database snapshots as backup.

5.2.8 Graph Representation

The Graph Representation component, depicted in Fig. 5.9, implements elastic databases for managing sequences of operations performed on time series during the forecasting process. Sequences are represented as graphs where nodes identify input and output time series while edges identify operators. *Key-value storage* databases are used to keep time series operation graphs and *relational databases* are used to store graph metadata. In order to enhance availability, databases are replicated on standby[3] instances following an *eventual consistency* pattern. Additional backup is provided with database snapshots preserved in *blob storage*.

5.2.9 RESTful Interfaces

The RESTful interfaces component, depicted in Fig. 5.10, implements a user interface as well as an Application Programming Interface (API) which provide tenants with access to ITSFCC by sending HTTP request messages via modern web browsers or mobile applications. Accesses to the user interface are synchronous while accesses to the API, implemented in a processing component, can be either synchronous or asynchronous. Both interfaces are configured on *block storage* (operating system, software, fast-access data storage).

5.2.10 Time Series Repository

The Time Series Repository component, depicted in Fig. 5.11, extracts from the data warehouse in the Multi-source Data Extraction component the time series to be forecast, and copies them into a reduced primary database for faster access. Time series values are put on an elastic *key-value storage* while

[3]A standby database is a synchronized copy of the primary database, usually kept for data protection. On the other hand, a replica database is a copy of the primary database optimized for reading operations, usually intended to enhance performance for read-heavy database workloads.

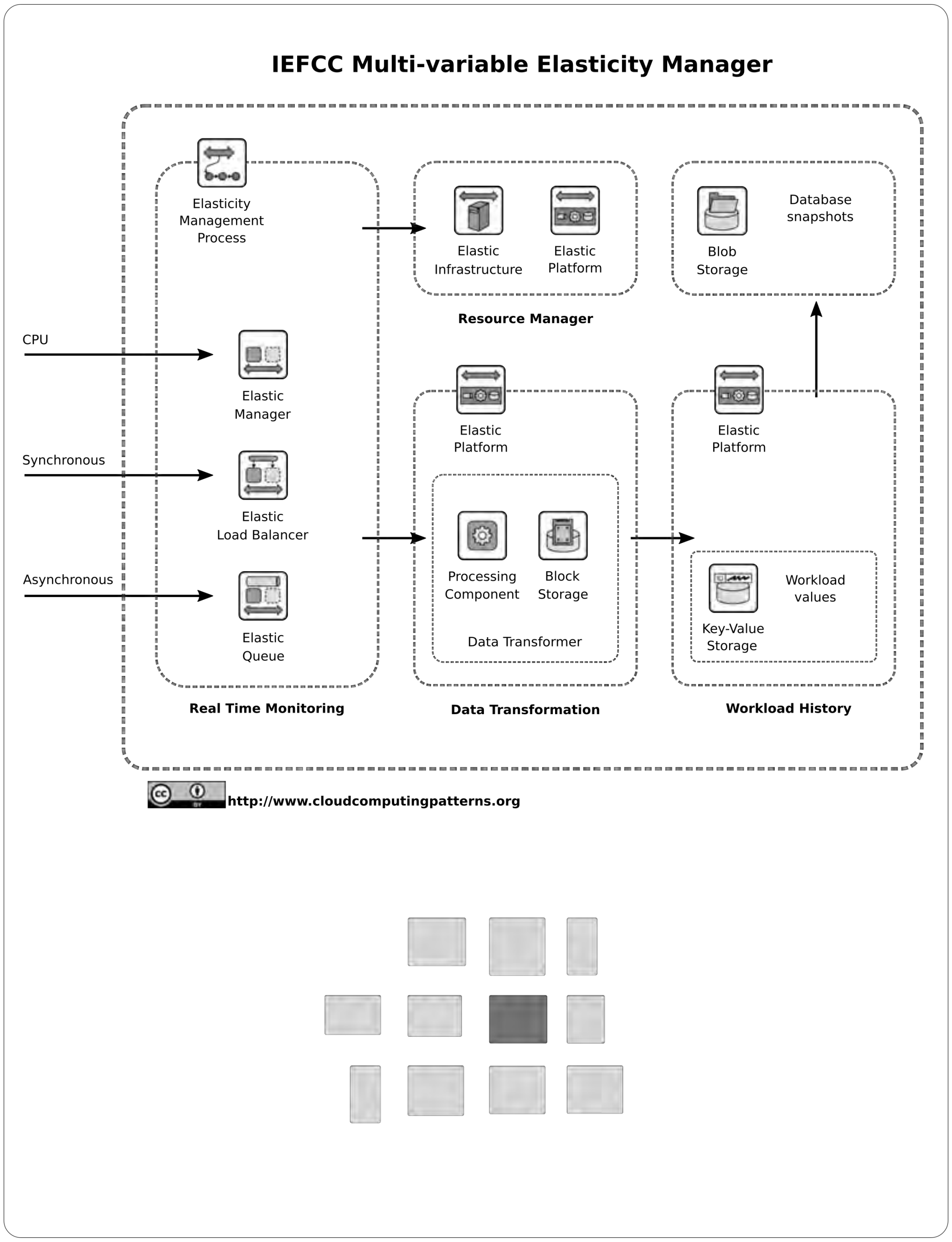

Figure 5.8: Multi-variable Elasticity Manager

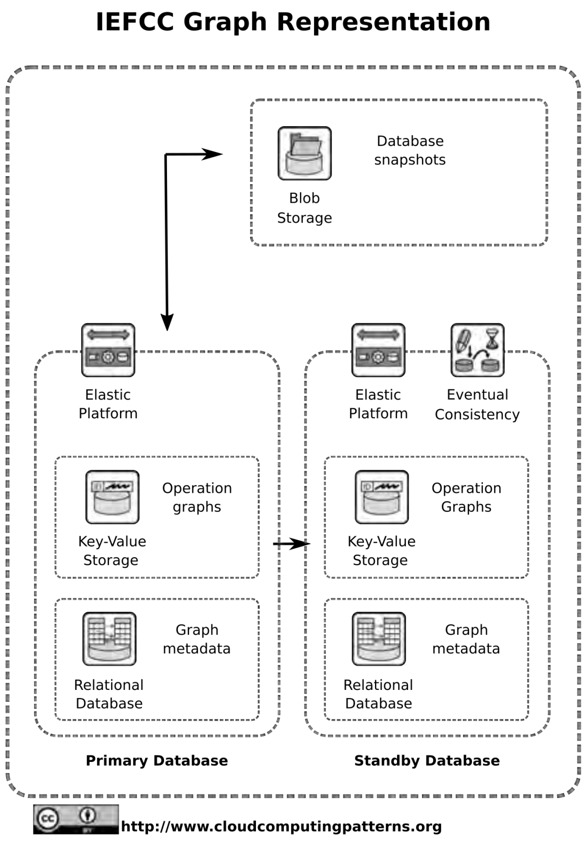

Figure 5.9: Graph Representation

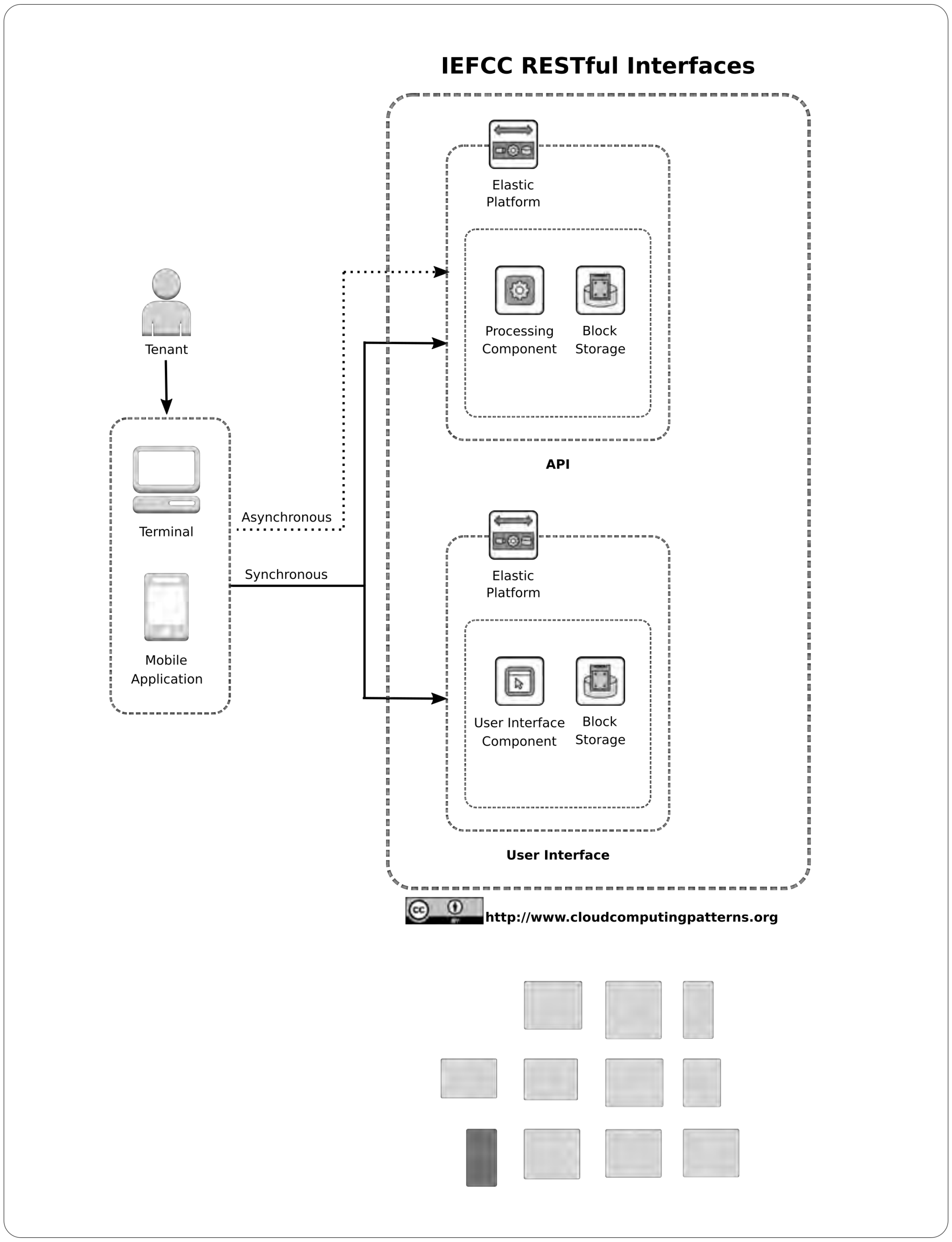

Figure 5.10: RESTful Interfaces

time series metadata is put on an elastic *relational database*. Backup for primary database is provided by an elastic standby database replicated in a different physical location, and by a set of elastic read-replica databases, assigned on a per-customer basis. All backup databases implement an *eventual consistency* pattern for ensuring tolerance to network partitioning and data availability over data consistency. Additional backup is provided by keeping snapshots of the primary database in *blob storage*.

5.2.11 Forecast Model Repository

The Forecast Model Repository component, depicted in Fig. 5.12, extracts from the data warehouse in the Multi-source Data Extraction component the context information elements of the time series to be forecast, and copies them into a reduced primary database for faster access. This primary database also contains a historic set of forecast models previously produced for all of the time series, including the one to be forecasted. Based on context similarity a forecast model in the repository is selected as an initial approximation for producing fast predictions for the time series, that are available while ITSFCC calculates the best model given the current information. Once the best model has been produced, it replaces the temporary model, and is stored along with its context information in the repository. The primary database comprises an elastic *key-value storage* for forecast model and context values, as well as an elastic *relational database* for storing metadata. It is replicated on an elastic standby database placed on a different physical location, which is updated following an *eventual consitency* pattern. Additional backup is provided by keeping snapshots of the primary database in *blob storage*. An elastic *processing component*, configured on *block storage* (operating system, software, fast-access data storage), implements the forecast model selector.

5.2.12 Batch Processing

The Batch Processing component, depicted in Fig. 5.13, processes the time series in the Time Series Repository to produce a forecast model based on the sequences of operations managed in the Graph Representation component. It implements at least three *processing components* on *elastic infrastructure* for data transformation, forecast model production, and model evaluation and benchmarking. All processing components are configured on *block storage*

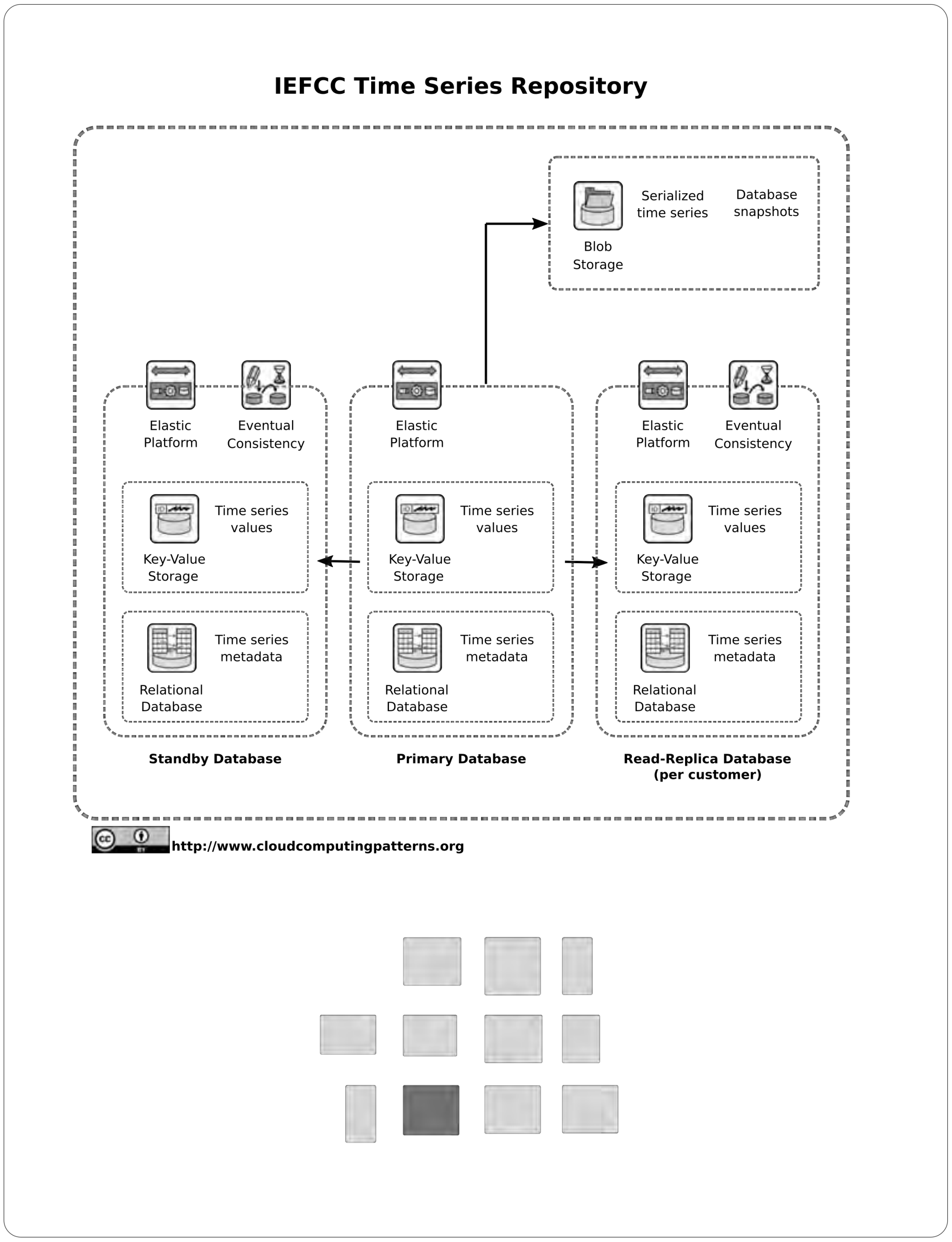

Figure 5.11: Time Series Repository

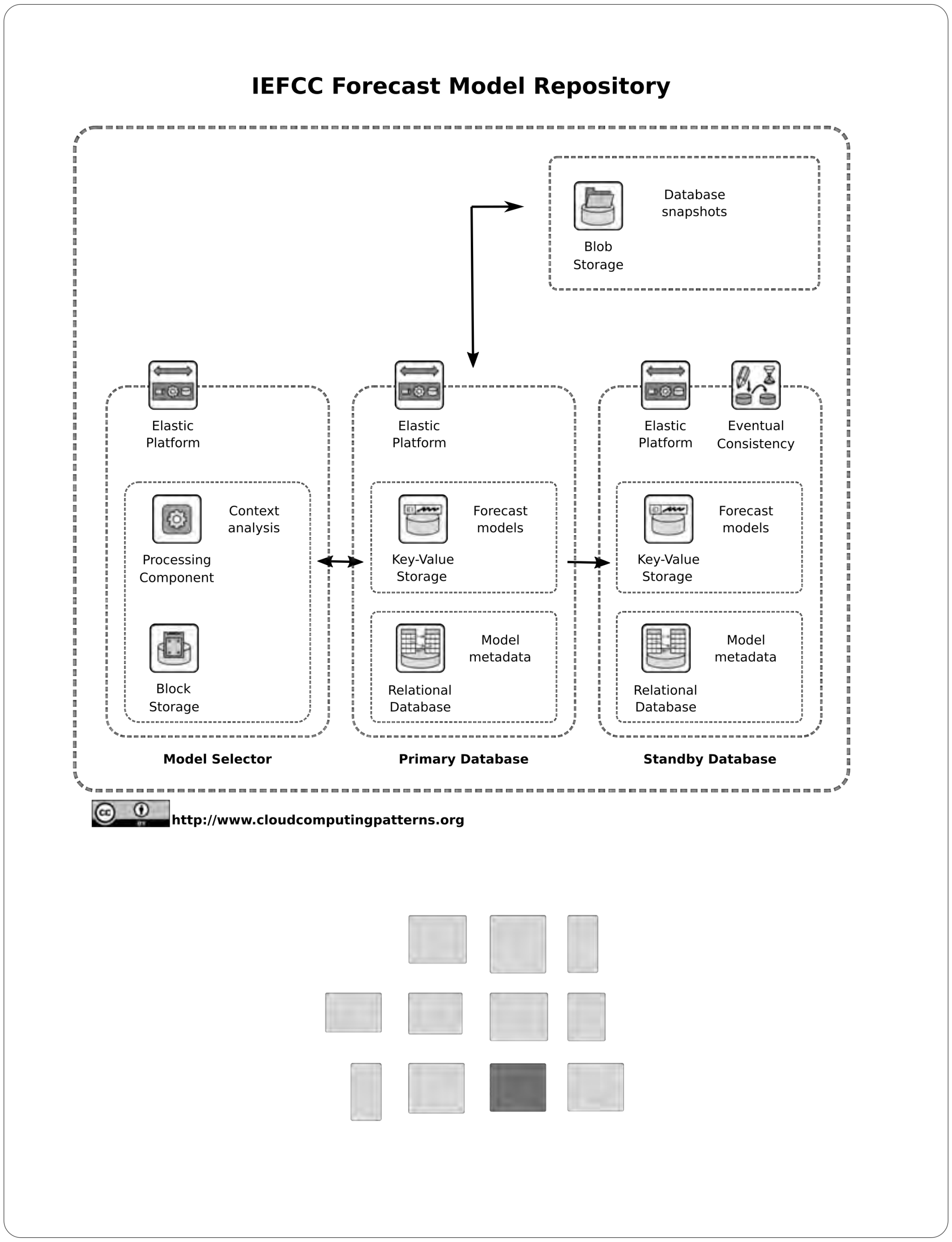

Figure 5.12: Forecast Model Repository

(operating system, software, fast-access data storage). Results from this application component are stored in the Forecast Model Repository.

5.2.13 Parallel Implementation

The Parallel Implementation component, depicted in Fig. 5.14, replicates the functionality of the Batch Processing component adding the deployment of a distributed computing platform to speed up the processing components. Therefore, it processes the time series in the Time Series Repository to produce a forecast model based on the sequences of operations managed in the Graph Representation component. It implements at least three *processing components* on *elastic infrastructure* for data transformation, forecast model production, and model evaluation and benchmarking. All processing components are configured on *block storage* (operating system, software, fast-access data storage) and implement a *map-reduce* pattern. Some Big Data processing frameworks suitable for implementing this component in ITSFCC are Apache Hadoop, Apache Spark, and Apache Flink. Results from this application component are stored in the Forecast Model Repository.

5.2.14 Pipes-and-Filters Processing

The Pipes-and-Filters Processing component, depicted in Fig. 5.15, integrates a set of the ITSFCC's components into an automated process that is executed in response to a tenant's request for producing forecasts for particular data sets. The process is event-driven and starts when the tenant requests ITSFCC operation, via the *RESTful interfaces* of the system, by uploading the data sets as serialized objects to the *blob storage* in the *Time Series Repository.* Context information elements for the data sets are automatically copied from the data warehouse in the Multi-source Data Extraction component and transferred to the *Forecast Model Repository* component. Then, either the *Batch Processing* component or the *Parallel Implementation* component produces and evaluates the appropriate forecast model based on a sequence of operations managed by the *Graph Representation* component. This sequence of operations can be overriden by the tenant, if requested, by uploading a customized operation graph in a YAML configuration file to a specific *blob storage* location in ITSFCC.

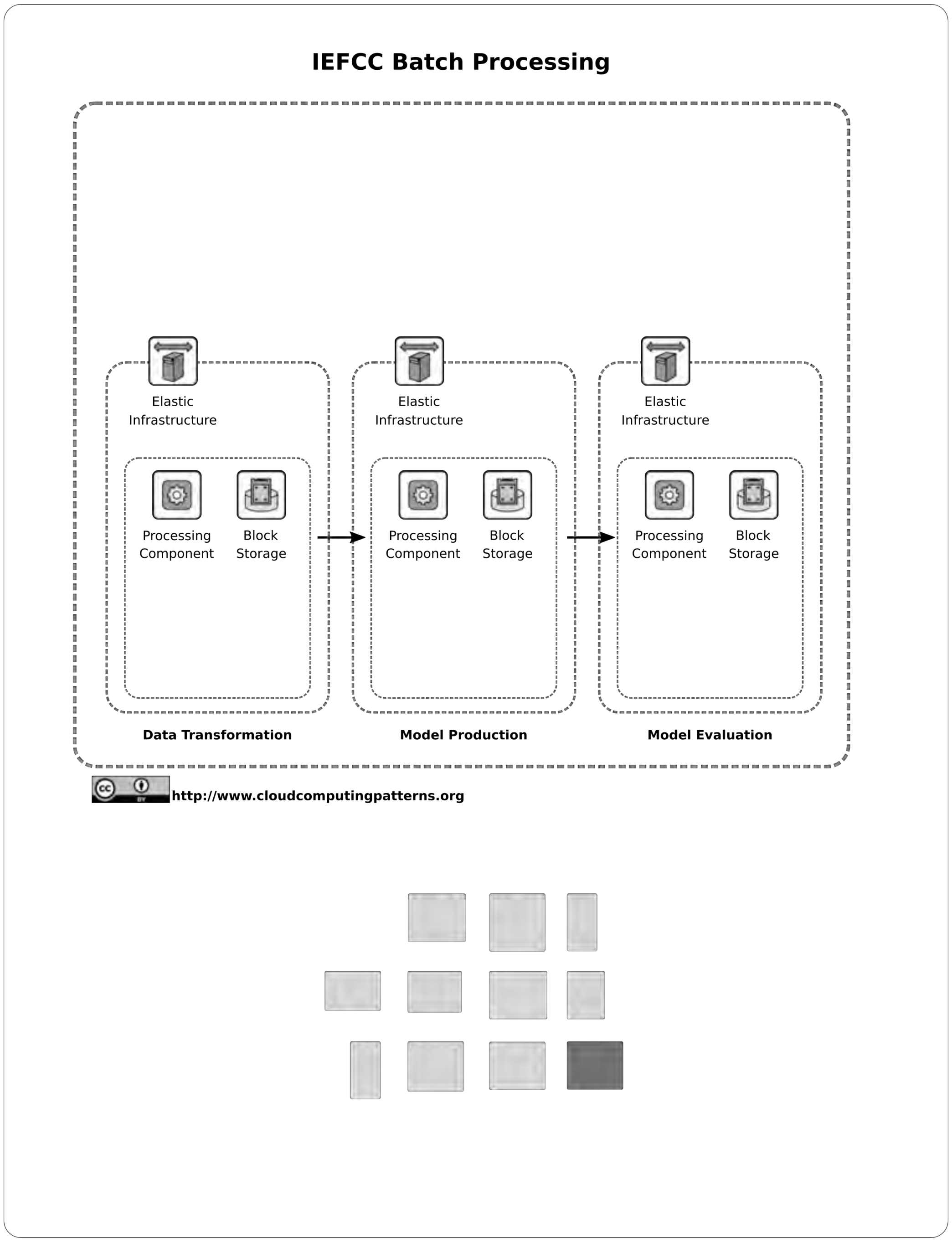

Figure 5.13: Batch Processing

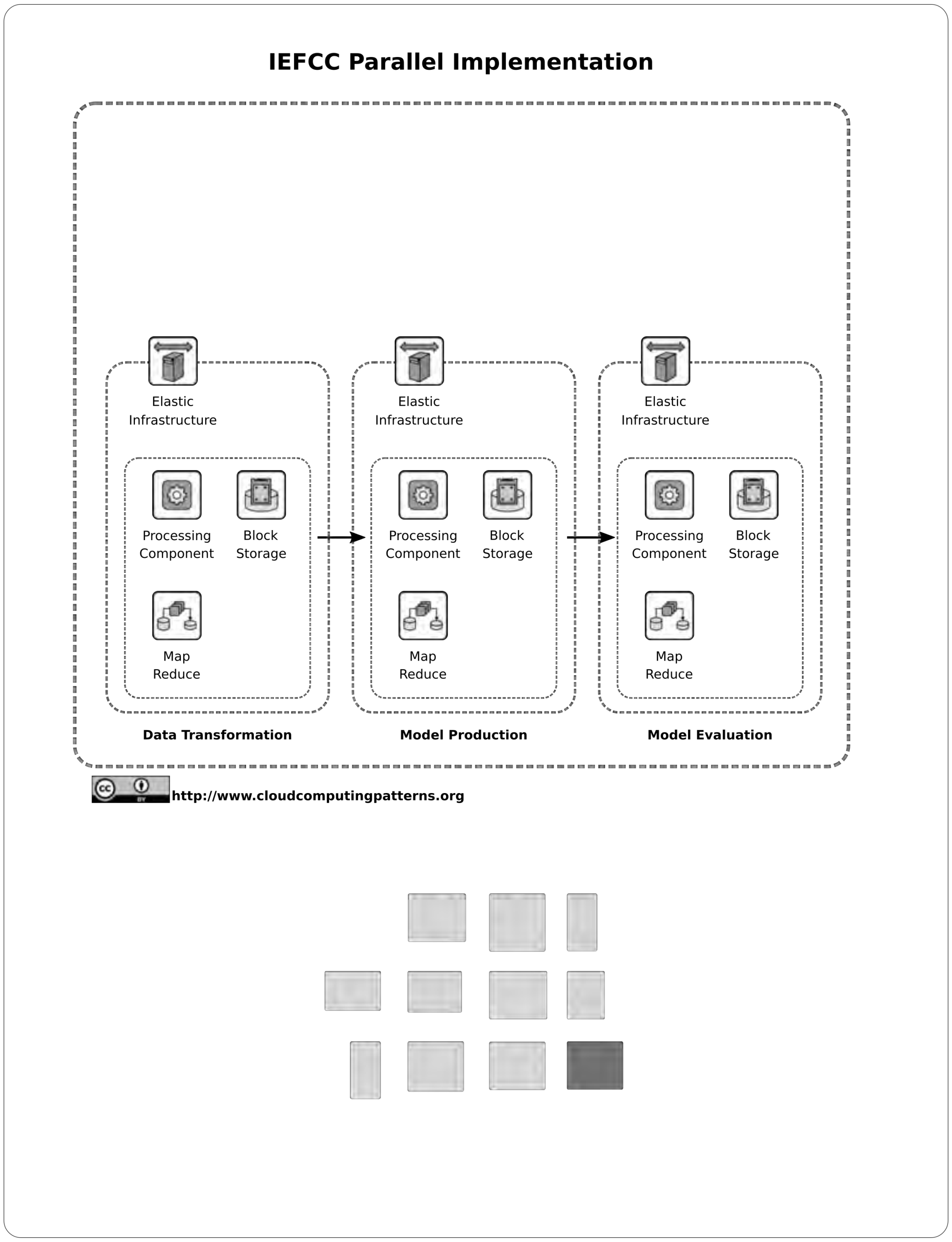

Figure 5.14: Parallel Implementation

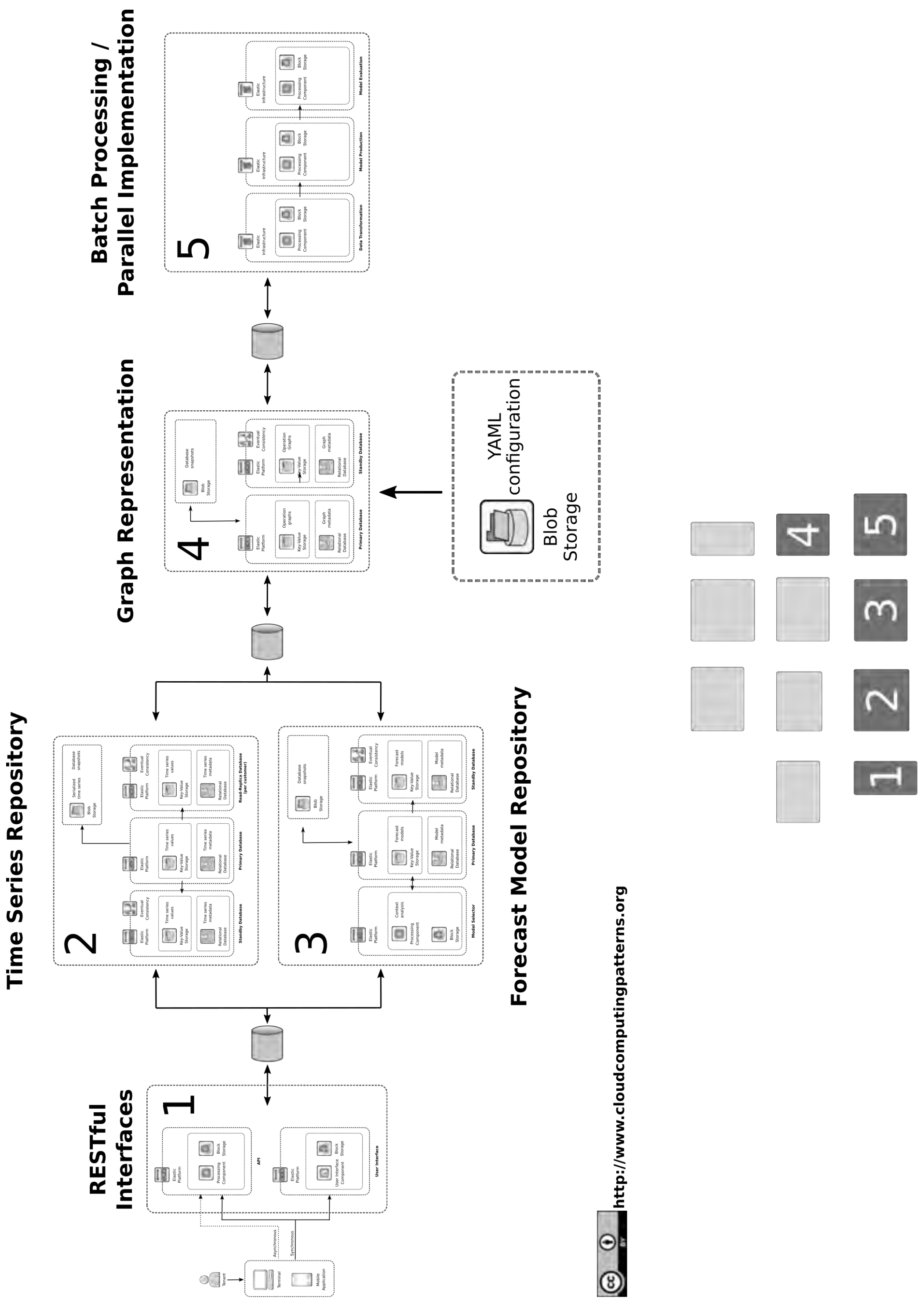
RESTful Interfaces
1
Time Series Repository
2
Forecast Model Repository
3
Graph Representation
4
Batch Processing / Parallel Implementation
5
YAML configuration
Blob Storage
http://www.cloudcomputingpatterns.org

Figure 5.15: Pipes-and-Filters Processing

5.2.15 Firewall Protection

The Firewall Protection component, depicted in Fig. 5.16, implements a Virtual Local Area Network comprised of at least three subnets or virtual network segments: a public subnet accesible from the internet, a private subnet accesible only from the virtual servers in the public subnet, and a second private subnet accessible only from the virtual servers in the first private subnet. Public subnet exposes user and application interfaces of ITSFCC to tenants. First private subnet allocates virtual servers where processing components are hosted. Second private subnet allocates virtual servers where databases are hosted. For development and delivery purposes, the first private subnet may also be accesible to ITSFCC staff via SSH from specific local IP addresses. As a consequence, ITSFCC's business logic and data layers are kept hidden from unauthorized access while the presentation layer is accessible to the world.

Summary

This chapter presented the detailed architecture characterization of ITSFCC, in compliance with the specifications covered in Chapter 2. Basic architecture documentation, including its identification and overview, the identification of the system's stakeholders, and the identification of their concerns was presented. A graphic concern map which groups ITSFCC application components into six categories and shows how the system concerns impact on each category was also presented. The overall architecture of the system as well as each one of its application components were presented as elements of the final architecture model, which conforms to a process-based architecture view and is governed by a block diagram model kind. The next chapter describes how this architecture characterization can be used as a source of important guidelines for prototyping ITSFCC on the most extensively used cloud computing platform nowadays: Amazon Web Services.

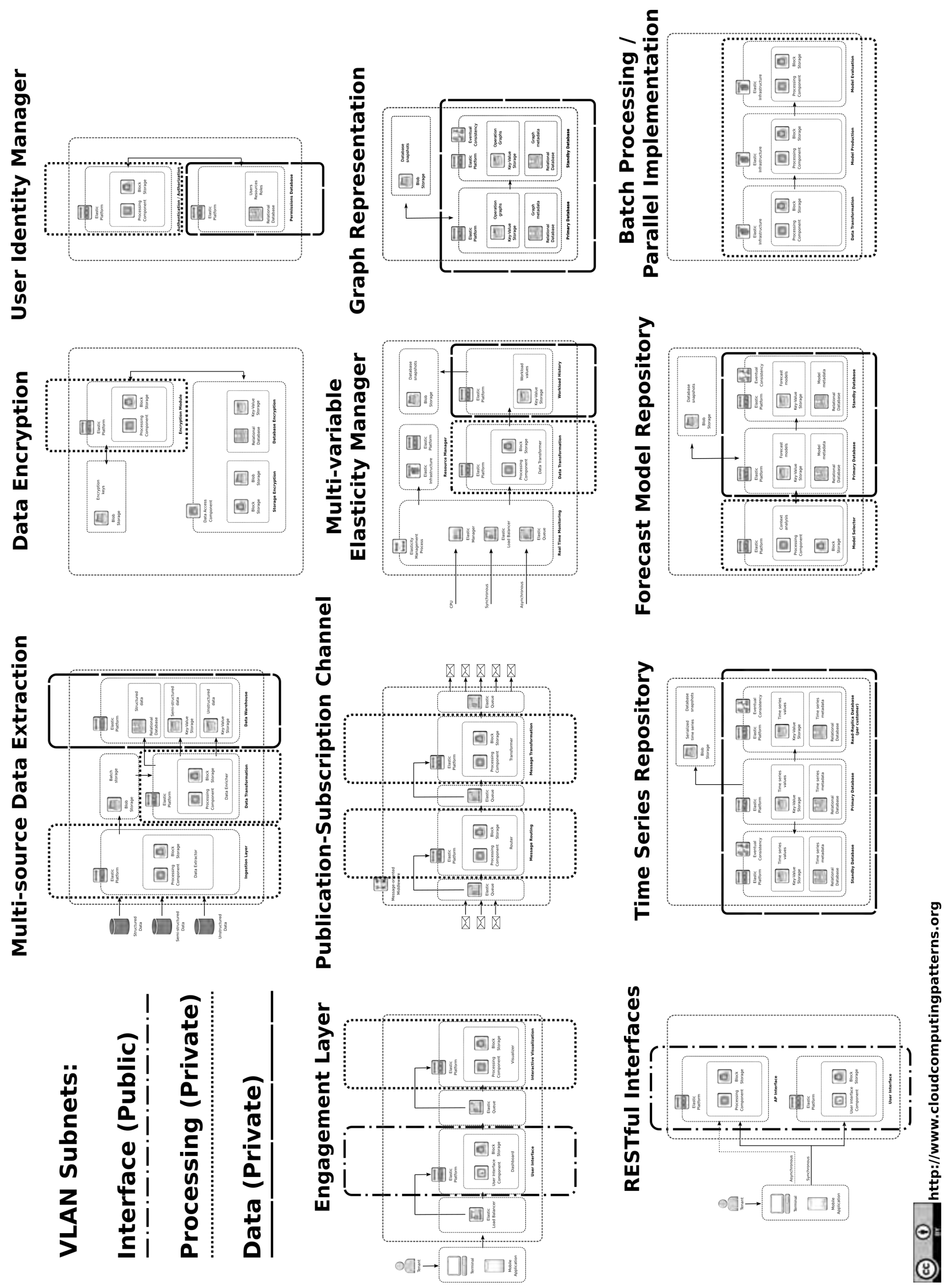
VLAN Subnets:
Interface (Public)
Processing (Private)
Data (Private)
Multi-source Data Extraction
Data Encryption
User Identity Manager
Engagement Layer
Publication-Subscription Channel
Multi-variable Elasticity Manager
Graph Representation
RESTful Interfaces
Time Series Repository
Forecast Model Repository
Batch Processing / Parallel Implementation
http://www.cloudcomputingpatterns.org

Figure 5.16: Firewall Protection

Chapter 6

ITSFCC on Amazon Web Services

This chapter departs from the detailed architecture characterization of ITSFCC presented in chapter 5 and maps it to the services offered by a specific public cloud provider. Amazon Web Services (AWS) was selected as the concrete cloud offering to land ITSFCC's abstract architecture on, because of the extension and maturity of its services, as well as for the facilities it offers to deploy cloud resources on a trial basis. Section 6.1 presents an overview of AWS and the services required to implement ITSFCC components in the AWS cloud. Section 6.2 presents a set of guides for implementing ITSFCC components in AWS. Each guide consists of a detailed architecture blueprint including numbered pointers that conduct a text-based explanation. Finally, section 6.3 summarizes a set of experiments deployed on AWS in order to validate the adequacy of specific services to the requirements placed by ITSFCC components.

6.1 Amazon Web Services Overview

Amazon Web Services started operations as a cloud computing provider in 2006 by releasing three products: the Simple Storage Service, the Simple Queue Service, and the Elastic Cloud Computing service. As of the first quarter of 2018, with a cloud offering that surpasses 90 services, AWS holds over a million active customers in 190 countries. With data center locations in the U.S., Canada, Brazil, India, South Korea, Singapore, Japan, China, Germany, France, England, Ireland, and Australia, AWS offers cloud resources allocated in 18 geographical regions and 54 availability zones [AWS, 2018]. AWS is the leader of the public cloud market with a 62 percent market share, distantly followed by Microsoft Azure (20 percent), and Google Cloud Platform (12 percent) [Novet, 2018].

The infrastructure of AWS is divided up into geographically diverse *regions* so as to provide speed and performance to globally distributed customers. Within a region, there are always multiple *availability zones*, or AZ, which represent geographically distinct -but still close- physical data centers. AZs have their own facilities and power source, so an event that might take a single AZ offline is unlikely to affect the other AZs in the region [Chan and Udell, 2017]. AWS cloud offering range from basic functionalities like compute, storage, databases, and networking services, to advanced features such as Internet of Things, Machine Learning, and Virtual Reality deployments. The following subsections, based on [AWS, 2018], briefly describe the services required to implement ITSFCC on AWS.

6.1.1 Compute

Services in this category provide secure and scalable compute capacity in the AWS cloud. A couple of them are included in the ITSFCC design:

- Amazon Elastic Cloud Computing (EC2). This service provides computing resources in the form of virtual server instances deployed on several operating systems. Instances are optimized to fit different use cases such as general purpose (M instances), burstable performance (T instances), compute (C instances), memory (X and R instances), accelerated computing (P, G, and F instances), and storage (H, I, and D instances).

- Elastic Load Balancing (ELB). This service automatically distributes incoming traffic to EC2 instances, containers, and IP addresses to make an application fault tolerant. It also makes the application highly available by monitoring target states and ensuring only healthy targets receive traffic.

6.1.2 Storage

Services in this category provide storage capacity in the AWS cloud. A couple of them are included in the ITSFCC design:

- Amazon Simple Storage Service (S3). This service provides *blob storage* with 99.999999999% durability of objects over a given year, which is equivalent to an average expected loss of 0.000000001% of the stored objects per year. This means, for instance, that from a set of 1,000,000 stored objects one of them is expected to be lost every 100,000 years.

- Amazon Elastic Block Storage (EBS). This service provides high-performance, low-latency, persistent *block storage* for EC2 instances. It is adequate for storing data in applications that benefit from fine tunning performance, such as Big Data analytics engines, relational and NoSQL databases, stream and log processing applications, and data warehousing applications.

6.1.3 Databases

Services in this category provide complete and diverse database functionalities in the AWS cloud. A couple of them are included in the ITSFCC design:

- Amazon Relational Database Service (RDS). This service provides complete *relational database* functionality with options to six widely used database engines: PostgreSQL, MySQL, MariaDB, Oracle, SQL Server, and Aurora.

- Amazon DynamoDB. This service provides NoSQL database functionality under two different models: *key-value* and document. It is adequate for applications that require consistent, single-digit millisecond latency at any scale.

6.1.4 Networking

Services in this category provide virtual networking appliances in the AWS cloud. Three of them are included in the ITSFCC design:

- Amazon Virtual Private Cloud (VPC). This service provides a logically isolated section of the AWS cloud where customers are able to deploy resources under a highly customized network. Subnets and firewalls can be defined inside the VPC to enforce the security of operations.

- Amazon Route 53. This service provides a highly available and scalable cloud Domain Name System (DNS) that routes end user requests to infrastructure running in AWS, such as EC2 instances, elastic load balancers or S3 buckets.

- Amazon API Gateway. This service allows developers to create, publish, maintain, monitor, and secure *application programming interfaces* at any scale. An API can be used as a “front door” to access data, business logic, or functionality from back-end services. API Gateway handles all the tasks involved in API calls processing, including traffic management, authorization and access control, monitoring, and version management.

6.1.5 Security

Services in this category provide security features in the AWS cloud. One of them is included in the ITSFCC design:

- AWS Identity and Access Management (IAM). This service provides secure access management to AWS services and resources through permissions granted to users in the form of security groups, roles, and policies.

6.1.6 Management

Services in this category provide management tools for resources deployed in the AWS cloud. One of them is included in the ITSFCC design:

- AWS Auto Scaling. This service monitors the usage of multiple AWS resources across multiple services and automatically adjusts them in order to maintain steady, predictable performance. Auto-scalable resources include EC2 instances, DynamoDB tables and indexes, and Aurora database replicas.

6.1.7 Application Integration

Services in this category allow fully managed communication among resources and applications deployed in the AWS cloud. A couple of them are included in the ITSFCC design:

- Amazon Simple Queue Service (SQS). It is a fully managed message queuing service that allows sending, storing, and receiving messages between application components at any volume, without losing messages or requiring other services to be always available.

- Amazon Simple Notification Service (SNS). It is a fully managed publication and subscription messaging service for coordinating the delivery of messages to subscribing HTTP endpoints and clients. Subscribers can be distributed systems, microservices, serverless applications, and mobile devices. It also allows to push messages directly into elastic queues from Amazon SQS in order to easily decouple and scale application components.

6.1.8 Analytics

Services in this category provide multiple analytics tools in the AWS cloud. A couple of them are included in the ITSFCC design:

- Amazon Elastic MapReduce (EMR). This service provides a managed Hadoop-based environment to process Big Data workloads across dynamically scalable EC2 instances. In addition to the components of the Hadoop ecosystem, like MapReduce, HBase, Hive, and Pig, other popular Big Data frameworks such as Spark, Presto, or Flink can be easily deployed in EMR.

- Amazon Redshift. This service provides a fully managed data warehouse service flexible enough to use standard SQL-based analytics as well as external business intelligence tools. By using sophisticated query optimization, columnar storage on high-performance disks, and massively parallel execution, Redshift is able to run complex analytic queries against petabytes of structured data, or exabytes of unstructured data in S3.

6.1.9 Machine Learning

Services in this category provide machine learning-based functionalities to applications deployed in the AWS cloud. One of them is included in the ITSFCC design:

- TensorFlow on AWS. This service provides a highly optimized environment for the execution of TensorFlow, a software library developed by Google Brain Team within Google's Machine Learning Intelligence research organization, for the purposes of conducting machine learning and deep neural network research [Zaccone, 2016]. An Amazon EC2 instance or cluster is fully provisioned for TensorFlow operation by unpacking a Deep Learning Amazon Machine Image (AMI). An AMI is a special virtual device that encapsulates all the information required to deploy a specific type of instance in the AWS cloud. It provides a template for the root volume of the instance, a set of permissions granted over the instance, as well as its block device mapping. Once the Deep Learning AMI is unpacked, machine learning-based applications can rapidly be developed, tested, deployed, and managed in the cloud. In the context of ITSFCC, TensorFlow is intended to provide a high-performance environment for running machine learning algorithms like Neural Networks, Support Vector Machines, and Nearest Neighbors as part of the forecast model production, evaluation and benchmarking within the Batch Processing application component.

6.2 Implementation Guides

This section presents a set of guides for implementing ITSFCC application components using the services in the AWS cloud above described. The guides are based on the general architecture block diagram for ITSFCC shown in Fig. 6.1, which was produced with Cloudcraft, a visual designer optimized for AWS that incorporates smart building components, enables real-time connection with AWS accounts, and is available as a web service [Cloudcraft, 2018]. Each guide comprises a blueprint that frames the application component inside the general diagram, along with numbered pointers that conduct the explanation given in the corresponding subsection.

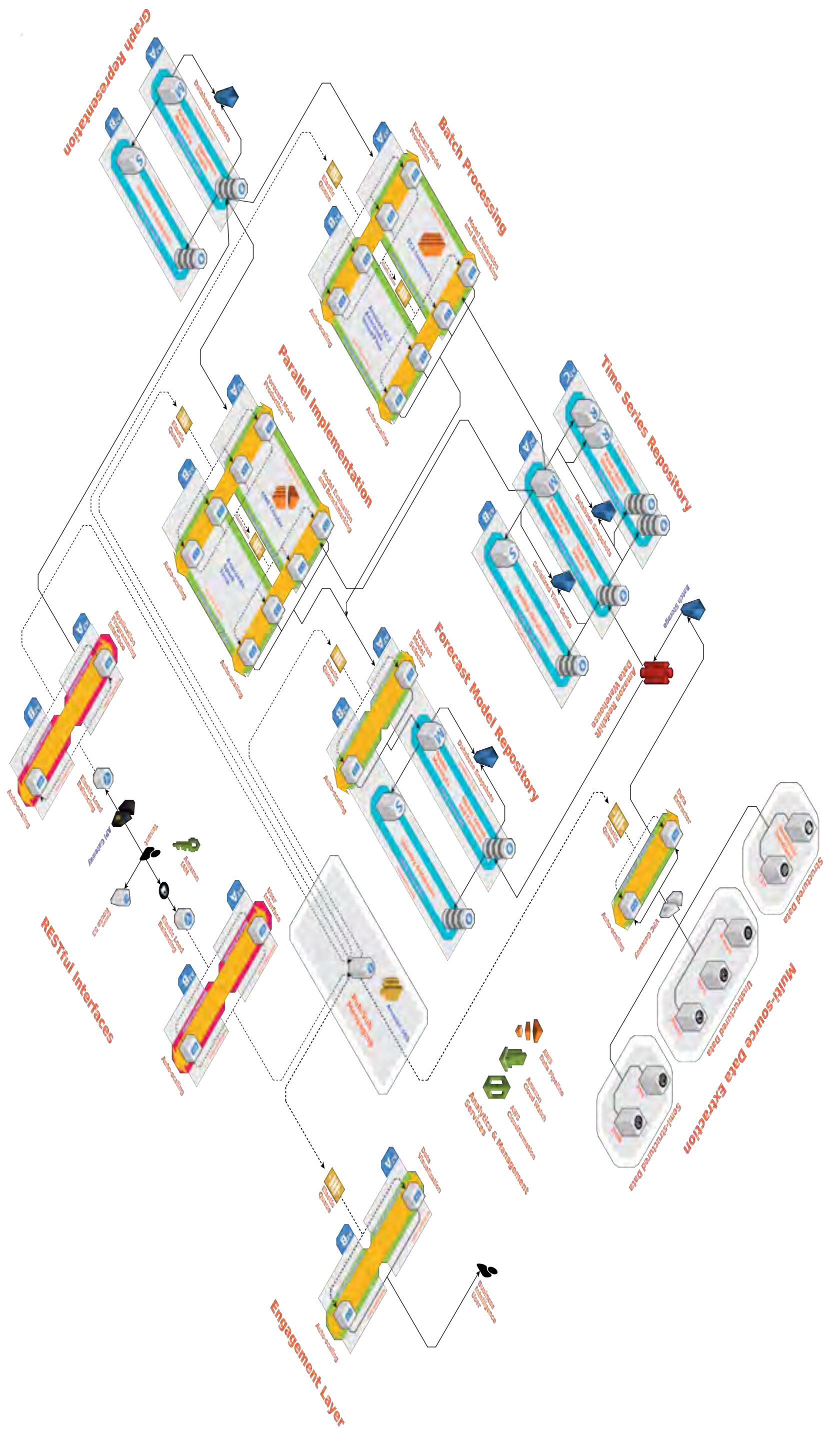

Figure 6.1: ITSFCC Architecture on Amazon Web Services

6.2.1 Application Components Included in AWS

The abstract ITSFCC architecture presented in Section 5.2 is comprised by 15 application components. A total of 6 application components are not considered in this concrete, provider-specific architecture because the services they address are already part of the core functionality of AWS. Following is a list with such components included in AWS:

- User Identity Manager. Amazon IAM performs the function of this application component via the definition of users and the authentication of their permissions in security groups, roles and policies.

- Data Encryption. Data stored in Amazon S3, Amazon DynamoDB, and Amazon RDS can be encrypted in the server side (the virtual servers and disks in the AWS cloud) at customer request, using the Amazon Key Management Service (KMS). If a higher level of control over the encryption process is required, client-side encryption is available via multiple tools. However it is important to bear in mind that encrypting data in the client-side involves a complex process of key management for the user.

- Multi-variable Elasticity Manager - The function of this application component is covered by AWS Auto Scaling, which dynamically scales outward or inward EC2 instances and DynamoDB tables according to pre-defined scaling policies and resource metrics provided by Amazon CloudWatch. Relational databases in Amazon RDS are horizontally scaled since their deployment by using read-replica instances of the primary database.

- Firewall Protection - The function of this application component is covered by Amazon VPC, via the definition of the appropriate subnets for the presentation (user interface), business logic (processing), and data (databases) layers. Once virtual servers have been allocated into subnets, a virtual firewall is defined with the security groups assigned to each EC2 instance in the VPC.

- Pipes-and-Filters Processing. Two services can be used to perform the functionality of this application component: AWS Data Pipeline is a service that allows to reliably process and move data between different AWS compute and storage services at specified intervals. Therefore it can be used to automatically manage the Pipes-and-Filters Processing component on a calendar basis. If for some reason the activation of the processing pipeline cannot be scheduled, AWS Lambda, an event-driven serverless computing platform, can be used to trigger it in response to a pre-defined event, such as the successful incoming of new data into the ITSFCC data lake.

- Distributed Application. Fig. 6.1 shows the general architecture block diagram of ITSFCC on AWS, where a total of eight application components are shown as loosely coupled elements that communicate through a Publication-Subscription Channel. All of the computing, database and storage resources are redundantly deployed on at least two Availability Zones, and all EC2 instances are placed in auto-scaling groups. As a result of these architecture features, all of the ITSFCC application components make use of multiple services in the AWS cloud to independently scale outward or inward when required.

The implementation guides for the remaining ITSFCC components are presented in the following subsections.

6.2.2 Multi-source Data Extraction

The Multi-source Data Extraction component, depicted in Fig. 6.2, extracts the time series values and contexts required for ITSFCC operation from multiple external sources. (1) Structured data coming from relational databases or CSV files, (2) unstructured data in the form of E-mail messages, text, and image files, and (3) semi-structured data coming from XML and JSON files enters ITSFCC through an Amazon VPC gateway (4) in the form of a data flow (5) pulled by the data extractor processing group (6). The data extractor is deployed on two general purpose M5-type Amazon EC2 instances which integrate an automatically scalable group inside the processing subnet, identified with the private IP address 10.0.4.0/24. Route tables in the processing subnet must declare the VPC gateway as the target for all Internet-routable traffic. All data in its original format is batch stored in Amazon S3 (7) to conform to the data lake pattern. An instance of Amazon Redshift (9) implements the data warehouse by pulling the required data from S3, transforming it in accordance to ITSFCC specifications, and storing it on NoSQL, columnar databases. The operation of this application component is started by a control flow (9) over the data extractor which comes in the form of request messages from an Amazon SQS elastic queue (10) fed by ITSFCC's management components (11).

6.2.3 Time Series Repository

The Time Series Repository component, depicted in Fig. 6.3, receives time series information (1) from the Amazon Redshift data warehouse in the Multi-source Data Extraction component. Based on the time series data format, Amazon Redshift sends the information directly from its columnar databases or transfers it from the S3 data lake. As long as there is no processing group in the Time Series Repository, this operation is started on-request by the application component that acts as customer, or in event-driven mode by the Pipes-and-Filters Processing component. Time series values are stored in Amazon DynamoDB using primary tables (2) for operation and standby tables (3) as backup. Time series values are also replicated on a per-customer basis on read-replica tables (4) for faster access. This replication procedure is also implemented for time series metadata on Amazon RDS, with primary (5), standby (6), and per-customer read-replica tables (7). In order to ensure high availability, primary, standby, and read-replica databases are placed in three different Availability Zones of the same AWS Region. For this reason, databases must also reside in different subnets identified with the private IPs 10.0.7.0/24, 10.0.8.0/24, and 10.0.9.0/24. For availability and performance purposes, serialized objects containing currently used time series are stored in S3 (8). Snapshots of DynamoDB and RDS databases are also stored in S3 (9). A faster but more expensive alternative for database backup can be achieved by replacing S3 storage with Amazon EBS volumes linked to the databases. Applications components that can activate the operation of the Time Series Repository are the Batch Processing component (10) and the Parallel Implementation component (11).

6.2.4 Forecast Model Repository

The Forecast Model Repository component, depicted in Fig. 6.4, receives time series context information (1) from the Amazon Redshift data warehouse in the Multi-source Data Extraction component. Based on the context data format, Amazon Redshift sends the information directly from its columnar databases or transfers it from the S3 data lake. Time series context values are stored in Amazon DynamoDB using primary tables (2) for

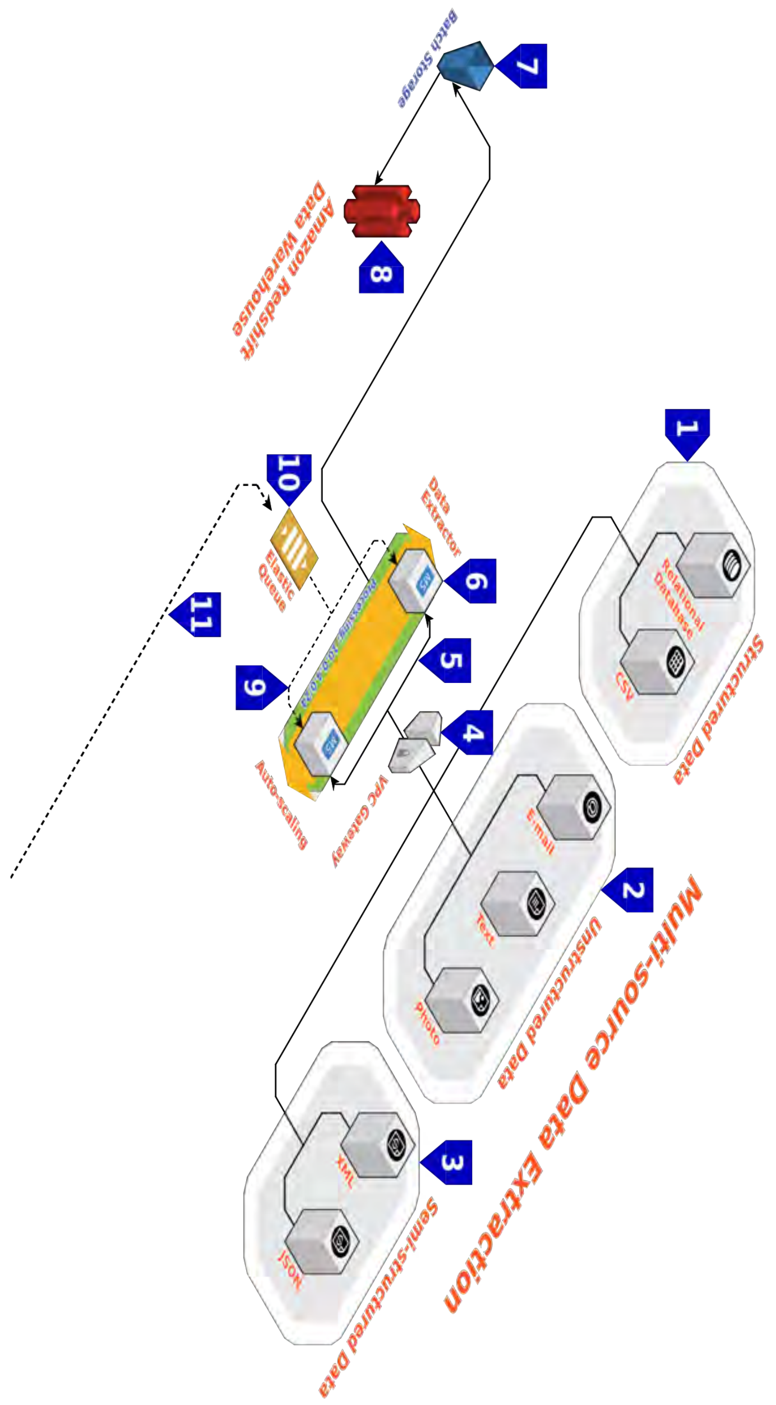

Figure 6.2: ITSFCC Multi-source Data Extraction on AWS

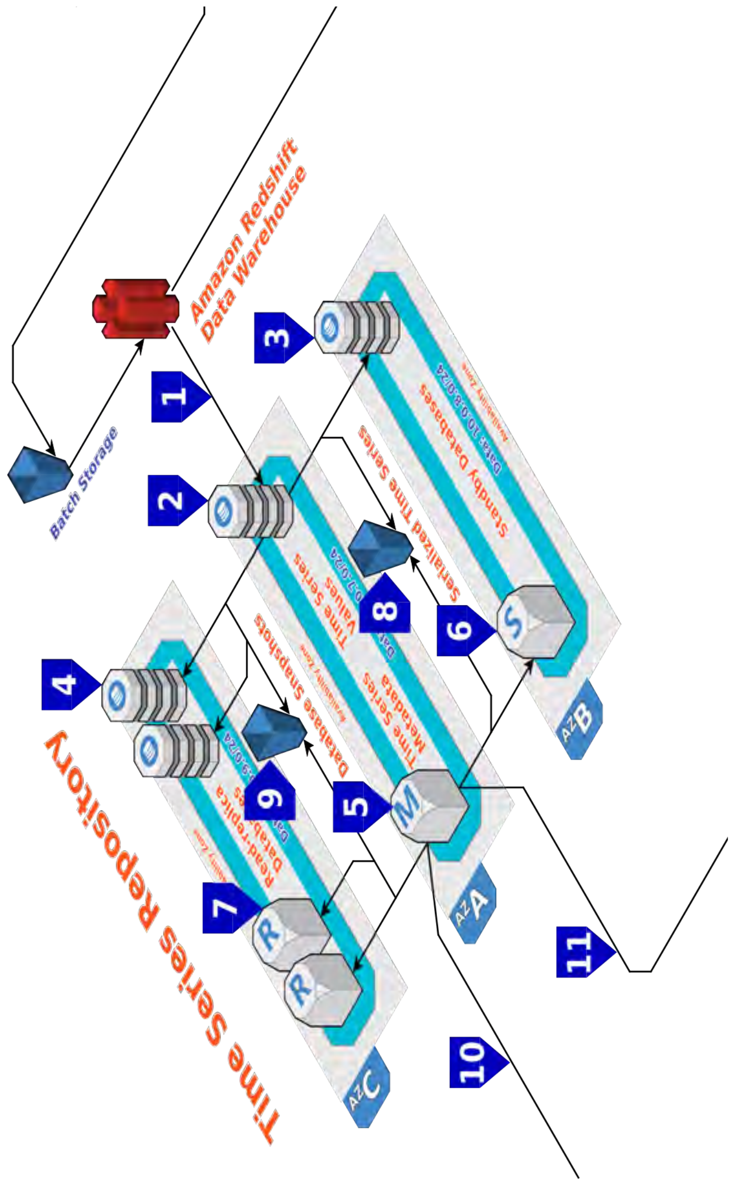

Figure 6.3: ITSFCC Time Series Repository on AWS

operation and standby tables (3) as backup. This replication procedure is also implemented for context metadata on Amazon RDS, with primary (4) and standby (5) tables. In order to ensure high availability, primary and standby databases are placed in two different Availability Zones of the same AWS Region. For this reason, databases must also reside in different subnets identified with the private IPs 10.0.7.0/24 and 10.0.8.0/24. For availability and performance purposes, snapshots of DynamoDB and RDS databases are stored in S3 (6). A faster but more expensive alternative for database backup can be achieved by replacing S3 storage with Amazon EBS volumes linked to the databases. When requested, this application component evaluates the context similarity between the current problem and the forecast models in the respository and selects the best previous model as an initial approximation for producing fast predictions. This forecast model selector is deployed on two compute-optimized C5-type Amazon EC2 instances (7) which integrate an automatically scalable group. As well as with databases, the EC2 instances are placed in different Availability Zones of the same AWS region for high availability. Therefore, they must reside in different subnets identified with the private IPs 10.0.4.0/24 and 10.0.5.0/24. Once a better forecast model for the present problem has been produced, it is stored in the repository (8) for further use. Application components that can produce a new forecast model to be stored in this component are the Batch Processing component and the Parallel Implementation component (9). The operation of the forecast model selector is started by a control flow which comes in the form of request messages from an Amazon SQS elastic queue fed by ITSFCC's management components (10).

6.2.5 Graph Representation

The Graph Representation component, depicted in Fig. 6.5, stores the sequences of operations to be performed on the time series in order to produce and evaluate a forecast model. Sequences of operations are coded as graphs which either are pre-stored as part of ITSFCC implementation or enter the system as a data flow sent by the application programming interface in the RESTful Interfaces component (1). Operation graphs are stored in Amazon DynamoDB using primary tables (2) for operation and standby tables (3) as backup. This replication procedure is also implemented for operation graph metadata on Amazon RDS, with primary (4) and standby (5) tables. In order to ensure high availability, primary and standby databases are

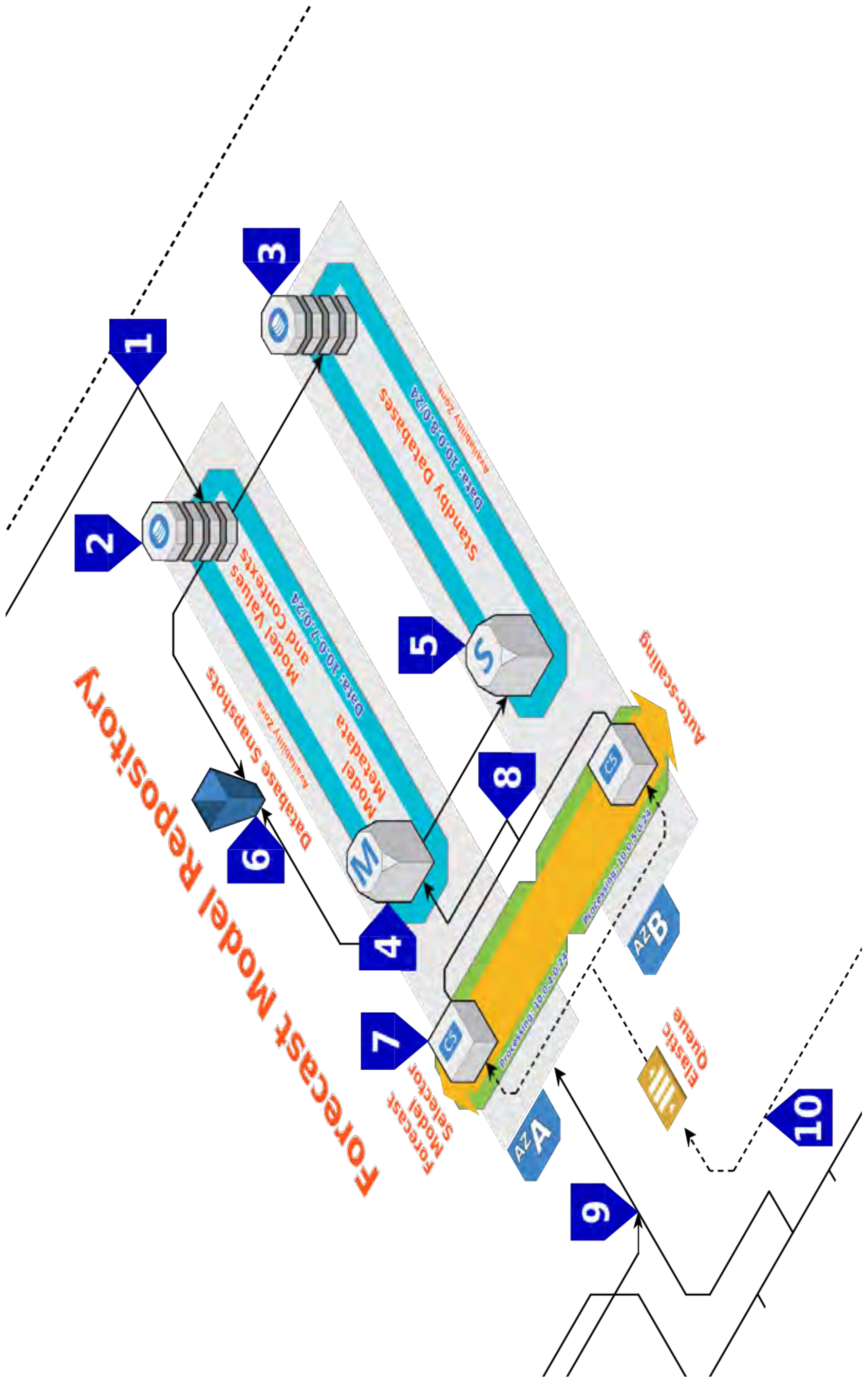

Figure 6.4: ITSFCC Forecast Model Repository on AWS

placed in two different Availability Zones of the same AWS Region. For this reason, databases must also reside in different subnets identified with the private IPs 10.0.7.0/24 and 10.0.8.0/24. For availability and performance purposes, snapshots of DynamoDB and RDS databases are stored in S3 (6). A faster but more expensive alternative for database backup can be achieved by replacing S3 storage with Amazon EBS volumes linked to the databases. Operation graphs are sent as a data flow to the Batch Processing component (7) and the Parallel Implementation component (8) for processing purposes.

6.2.6 Batch Processing

The Batch Processing component, depicted in Fig. 6.6, is started by a control flow which comes in the form of request messages from an Amazon SQS elastic queue fed by ITSFCC's management components (1). Time series data (2) provided by the Time Series Repository is used to produce a forecast model in an automatically scalable processing group, integrated by four accelerated-computing G3-type EC2 instances, placed on pairs in two different Availability Zones (3), (4) of the same AWS Region for high availability. A second automatically scalable processing group, comprised of four compute-optimized C5-type EC2 instances, evaluates the forecast model and runs benchmarking tests in response to requests messages coming from an Amazon SQS elastic queue (5) fed by the forecast model production processing group. EC2 instances in the second group are also placed in pairs on two Availability Zones (6), (7) of the same AWS Region for high availability. As long as EC2 instances are deployed in different AZs, they must reside in different processing subnets identified with the private IPs 10.0.4.0/24 and 10.0.5.0/24. All of the EC2 instances are provisioned with Anaconda, a widely used free Python distribution with almost 200 packages for science, math, engineering, and data analysis [Nandi, 2015]. Python was selected as the fundamental programming language for ITSFCC development because it is highly expressive and easy-to-learn, it includes a broad range of libraries developed by a huge community of developers, it provides an efficient interactive programming mode, and its structure and concept make it is easier to write and maintain large programs than in any other scripting language. According to [Sethi, 2017], these features make Python the best programming language for cloud native microservices development. The EC2 instances in the forecast model production group are also provisioned with the deep learn-

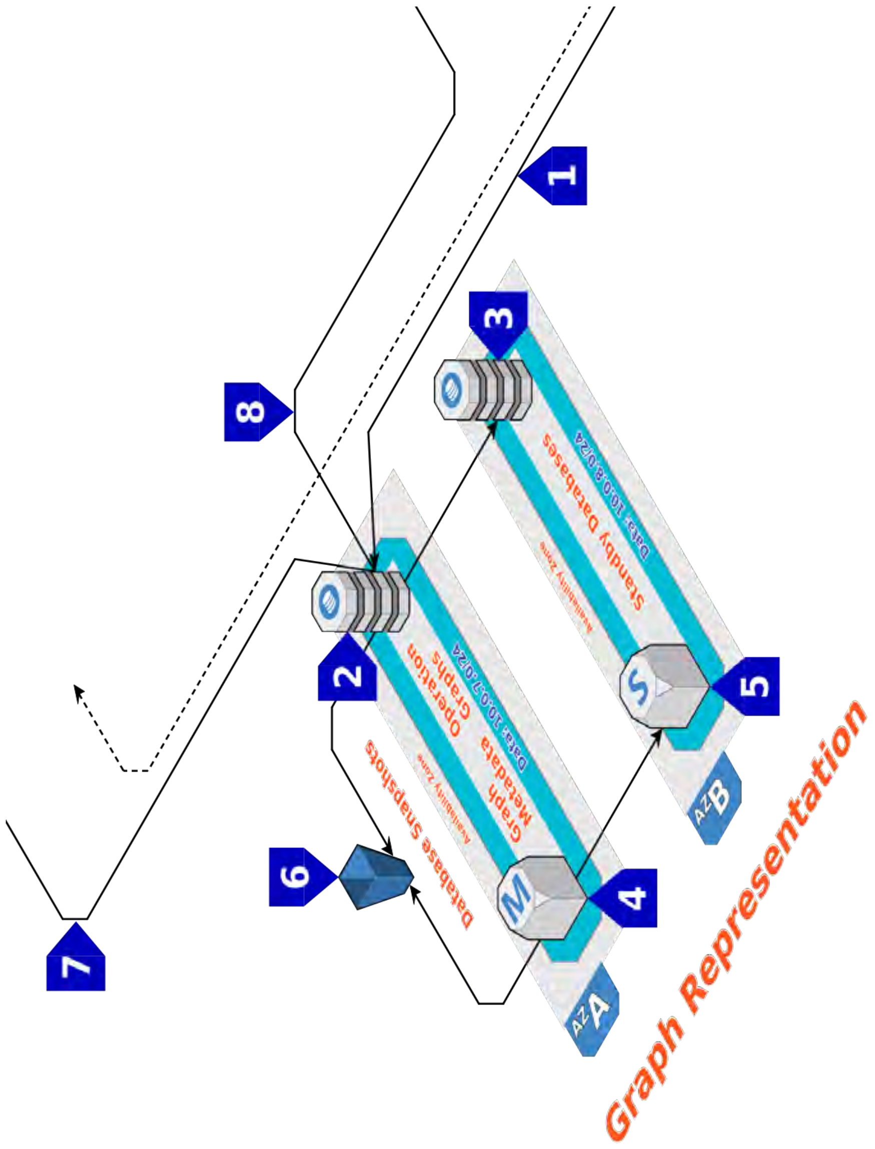

Figure 6.5: ITSFCC Graph Representation on AWS

ing framework TensorFlow. Once produced and evaluated, forecast models are stored in the Forecast Model Repository (8).

6.2.7 Parallel Implementation

The Parallel Implementation component, depicted in Fig. 6.7, is started by request messages from an Amazon SQS elastic queue fed by ITSFCC's management components (1). Time series data (2) provided by the Time Series Repository is used to produce a forecast model in an automatically scalable processing group, integrated by four compute-optimized C5-type EC2 instances, placed on pairs in two different Availability Zones (3), (4) of the same AWS Region for high availability. A second automatically scalable processing group, comprised of four compute-optimized C5-type EC2 instances, evaluates the forecast model and runs benchmarking tests in response to requests messages coming from an Amazon SQS elastic queue (5) fed by the forecast model production processing group. EC2 instances in the second group are also placed in pairs on two Availability Zones (6), (7) of the same AWS Region for high availability. As long as EC2 instances are deployed in different AZs, they must reside in different processing subnets identified with the private IPs 10.0.4.0/24 and 10.0.5.0/24. All of the EC2 instances in the application component are provisioned with Anaconda, while the EC2 instances in the forecast model production group are also provisioned with Apache Spark, an open-source powerful distributed querying and processing engine that provides the flexibility and extensibility of MapReduce but at significantly higher speeds: up to 100 times faster than Apache Hadoop when data is stored in memory and up to 10 times when accessing disk. Apache Spark was selected for the Parallel Implementation component of ITSFCC not only because of its outstanding performance features, but also for the vast community it encompasses, with more than 1,000 contributors from 250+ organizations and with 300,000+ Apache Spark Meetup community members in more than 570 locations worldwide [Drabas and Lee, 2017]. Apache Flink, a newer and more efficient distributed computing framework than Apache Spark, but at present time used by a significantly smaller community, can be considered as a suitable alternative for future development of this component. Once produced and evaluated, forecast models are stored in the Forecast Model Repository (8).

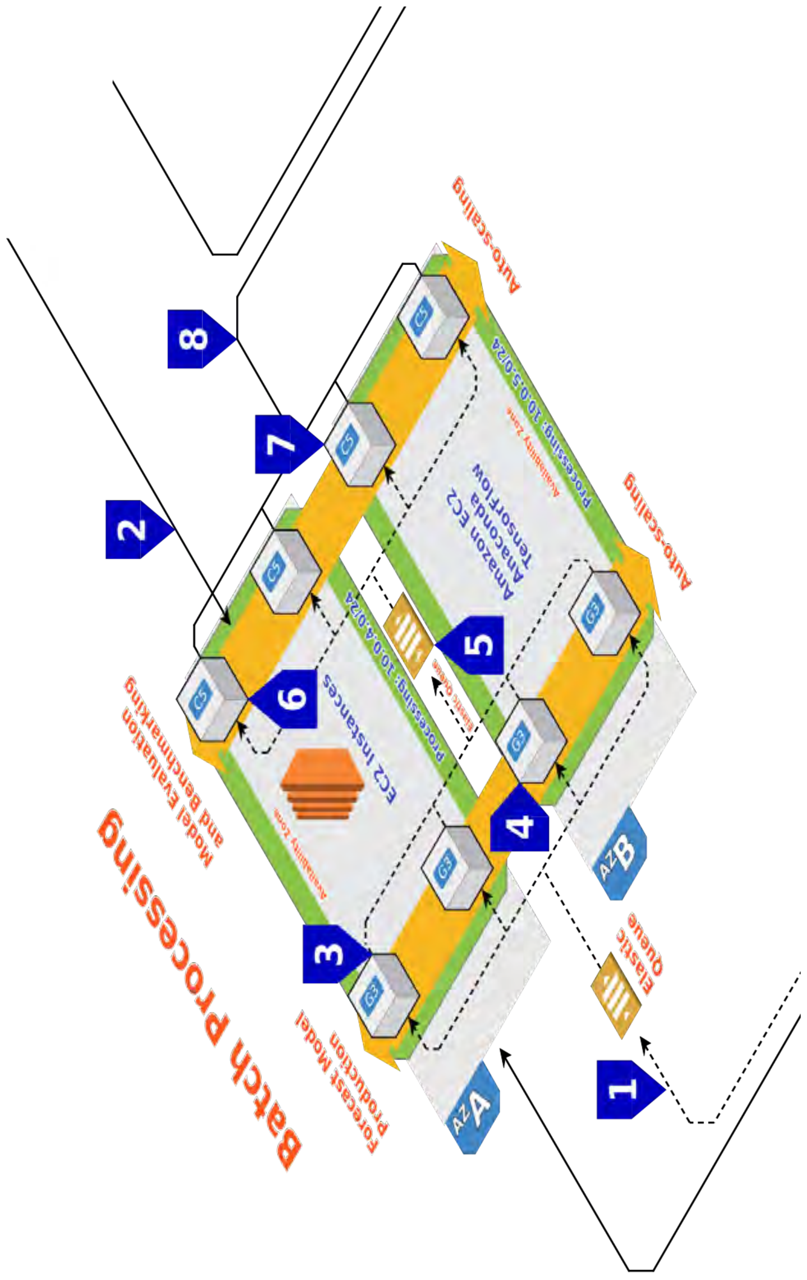

Figure 6.6: ITSFCC Batch Processing on AWS

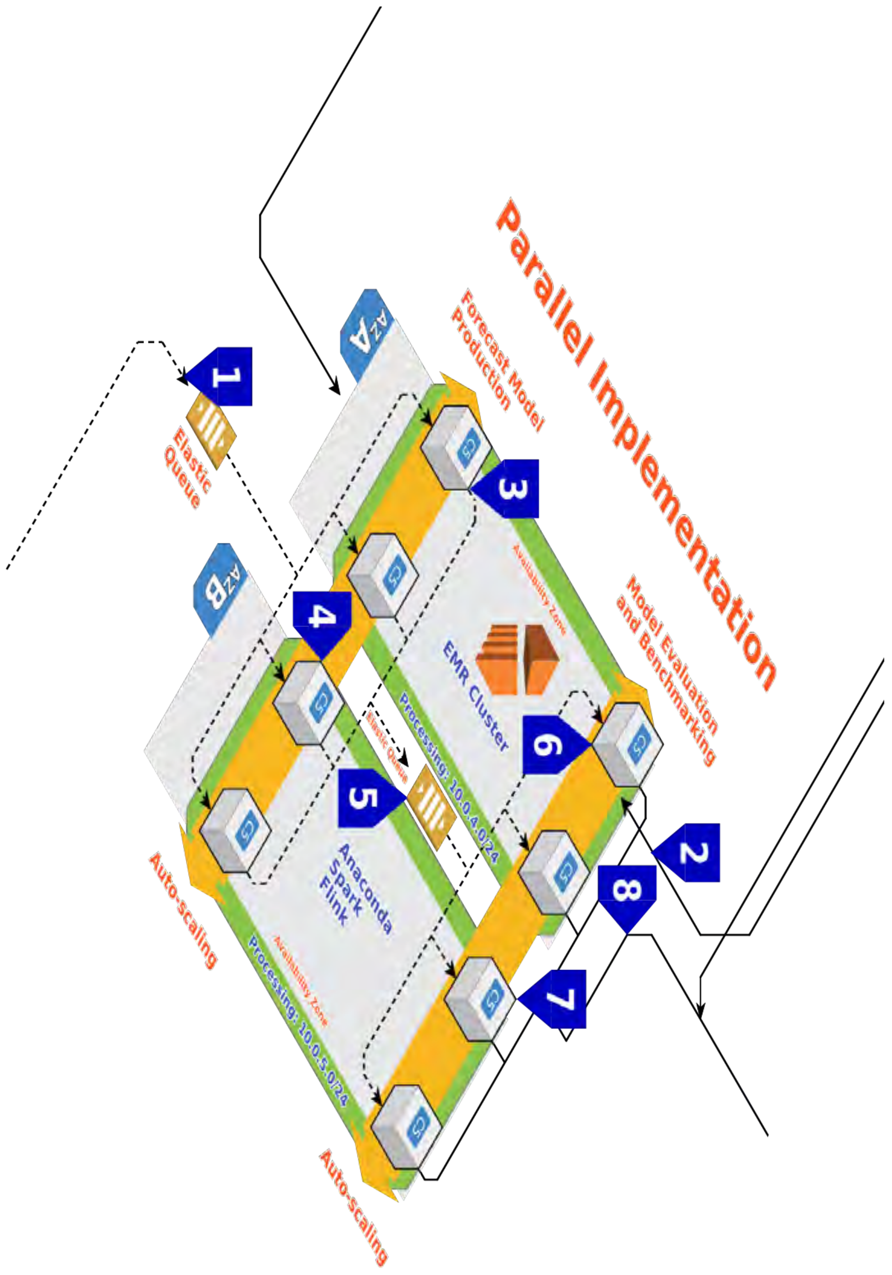

Figure 6.7: ITSFCC Parallel Implementation on AWS

6.2.8 RESTful Interfaces

The RESTful Interfaces[1] component, depicted in Fig. 6.8, handles DNS requests placed by a customer or tenant (1) using the cloud DNS Amazon Route53 (2). Incoming traffic is routed to infrastructure running in AWS after the tenant's credentials are validated by Amazon IAM (3). HTTP requests to ITSFCC user interface are handled by a load balancer from Amazon Elastic Load Balancing (4), which automatically distributes traffic to the web server group. This is an auto-scaling group integrated by two general purpose M5-type EC2 instances, placed in different Availability Zones (5), (6) of the same AWS Region for high availability. As long as EC2 instances are deployed in different AZs, they must reside in different interface subnets identified with the private IPs 10.0.1.0/24 and 10.0.2.0/24. Request messages coming out of the user interface group are passed as a control flow (7) to the Publication-Subscription channel. If the tenant's requests are for the ITSFCC application programming interface, they are routed to an instance of Amazon API Gateway (8) which is able to directly pass each request to the appropriate ITSFCC application component, as long as all of them are exposed in REST style. However, in order to provide a more complex utilization scheme to the user of the API, a processing group is assigned to design elaborate sequences of operations and to route them to the backend components via the Publication-Subscription Channel. Requests processed in this way are handled by a load balancer from Amazon Elastic Load Balancing (9), which automatically distributes traffic to the API processing group. This is an auto-scaling group integrated by two general purpose M5-type EC2 instances, placed in different Availability Zones (10), (11) of the same AWS Region for high availability. As long as EC2 instances are deployed in different AZs, they must reside in different interface subnets identified with the private IPs 10.0.1.0/24 and 10.0.2.0/24. Request messages coming out of the application programming interface group are passed as a control flow (12) to the Publication-Subscription channel. In addition, the tenant can use the application programming interface to pass data elements (13) to backend components in ITSFCC, such as configuration files describing customized se-

[1]ITSFCC's user interface and application programming interface will have to be exposed as RESTful web services, which means they operate in response to HTTP request messages composed by a HTTP method (i.e. POST, GET, PUT, DELETE), the URI that locates the service over the network, and the request body and header, which describe the data required for the operation.

quences of operations to be stored in the Graph Representation component.

6.2.9 Engagement Layer

The Engagement Layer component, depicted in Fig. 6.9, is started by a control flow which comes in the form of request messages from an Amazon SQS elastic queue fed by ITSFCC's management components (1). Interactive visualization of forecasting results is provided by an automatically scalable processing group, integrated by two compute-optimized C4-type EC2 instances (2) placed in different Availability Zones of the same AWS Region for high availability. As long as EC2 instances are deployed in different AZs, they must reside in different processing subnets identified with the private IPs 10.0.4.0/24 and 10.0.5.0/24. Forecasting visualization is sent as a data flow (3) to the Business Intelligence User (4) that requested it, via a user interface instance handled by the RESTful Interfaces component.

6.2.10 Publication-Subscription Channel

The Publication-Subscription Channel component, depicted in Fig. 6.10, implements an Amazon Simple Notification Service instance (1) to provide a Pub/Sub Messaging channel to the system. Requests for ITSFCC operation enter the Pub/Sub channel as control flows coming from the user interface group (2) or the application programming interface group (3). Acording to a set of previously defined topics, notifications in the form of HTTP request messages are sent to the application components that are subscribed to the Pub/Sub channel: the Multi-source Data Extraction (4), the Forecast Model Repository (5), the Parallel Implementation (6), the Batch Processing (7), and the Engagement Layer component (8). The figure also depicts three additional Amazon Web Services, not included in the ITSFCC architecture yet, that must be considered in the future as a complement to the Pub/Sub channel for ITSFCC management. AWS Data Pipeline (8) is a service that allows to reliably process and move data between different AWS compute and storage services at specified intervals. It is intended to automatically manage the Pipes-and-Filters Processing component on a calendar basis. Amazon Cloud Watch (9) is a cheap and easy-to-use centralized monitoring service that provides a variety of features such as alerts, logging, notifications, and custom metrics [Wadia, 2016]. As a part of ITSFCC management it can, for

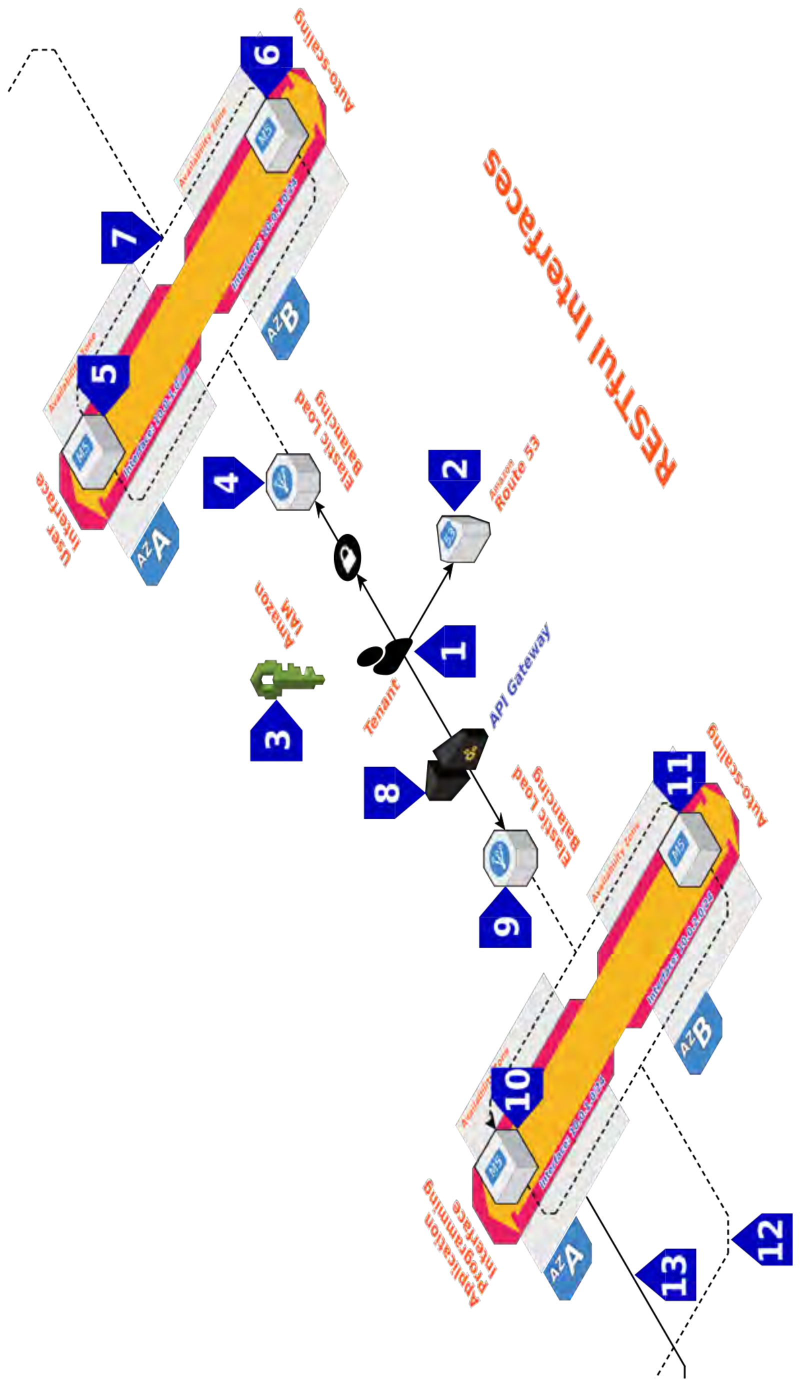

Figure 6.8: ITSFCC RESTful Interfaces on AWS

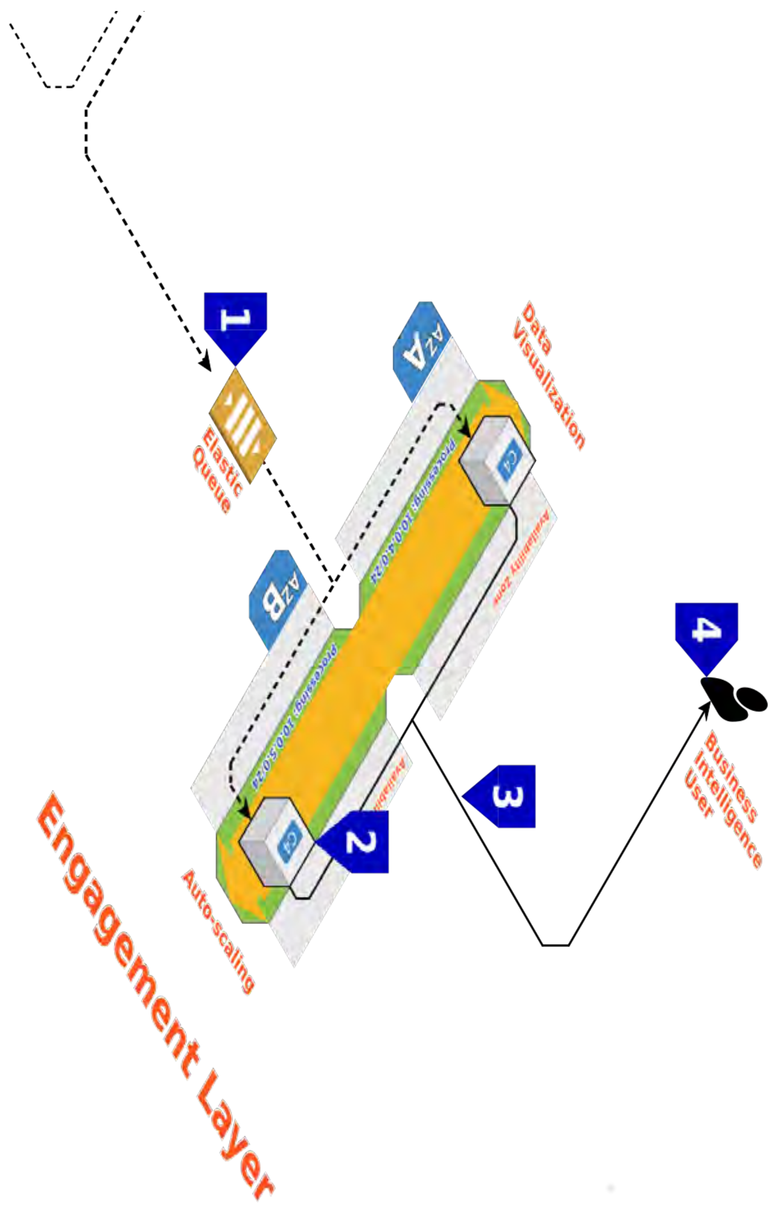

Figure 6.9: ITSFCC Engagement Layer on AWS

instance, invoke customized, automated workflows that exchange workloads between the Batch Processing and the Parallel Implementation components, in order to balance their performance metrics. Finally, AWS CloudFormation is a service that allows to provision and manage a collection of AWS resources, also referred to as stack, in an automated and repeatable fashion [Chan and Udell, 2017]. It is intended to speed up the deployment of ITSFCC infrastructure after shutdown periods caused by failure or maintenance.

6.3 Summary of Experiments on the AWS Cloud

This section describes a set of basic and intermediate-level experiments that were conducted on the AWS cloud as part of the ITSFCC architecture design. Although the fundamental goal of this work is to propose an abstract architecture for the software application, mapping such abstract design to the specific features of a concrete offering demands the deployment and test of as much services as possible. This is essential to get a better understanding of the available resources, their structure, operation logic, and syntax, as well as to validate their capabilities working as standalone units or as part of integrated workflows. The experiments performed on AWS for ITSFCC architecture design covered the following topics:

- Storage, databases, and pipelines.
- Deep Learning AMI.
- Spark on Amazon EMR.
- Virtual Private Cloud.
- Bokeh server on a customized AMI.

The following subsections describe the procedure and the extent of each experiment.

6.3.1 Storage, Databases, and Pipelines

Sample time series related to the energy domain were serialized into the Pandas dataframe format and then programatically uploaded to S3, by running

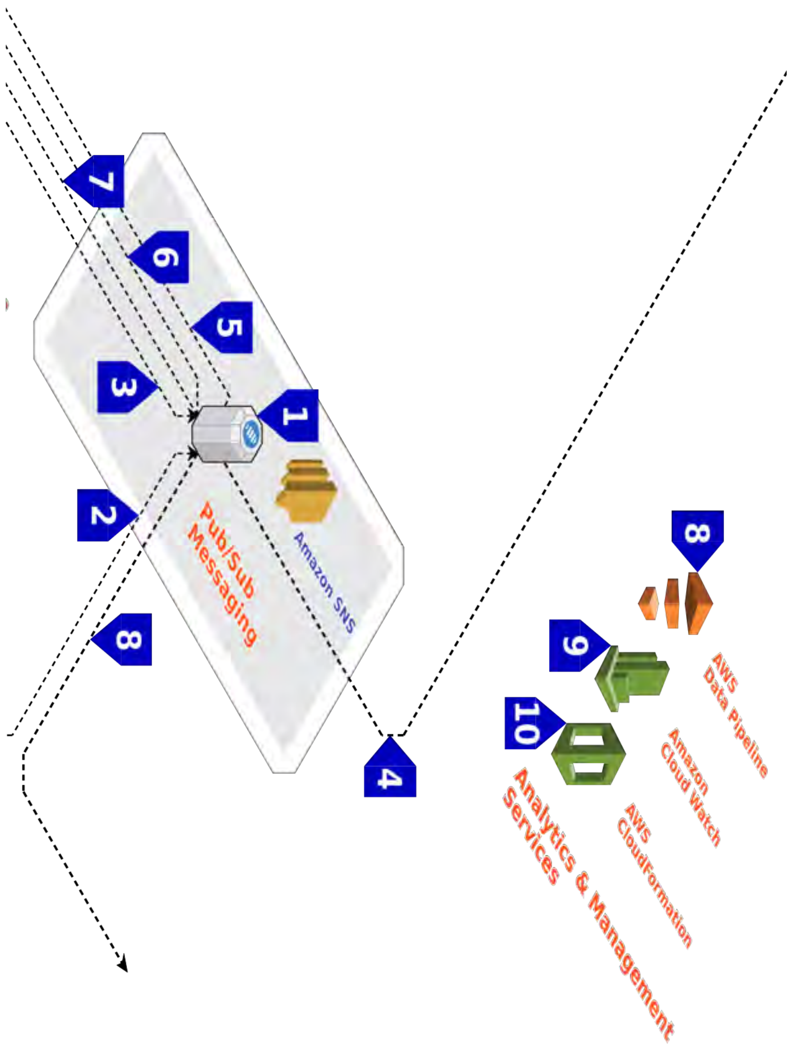

Figure 6.10: ITSFCC Management Components on AWS

simple applications in a T2-type EC2 instance. The applications leveraged **boto3**, a Software Development Kit (SDK) for Python that allows communication with the AWS application programming interface. In addition, a DynamoDB key-value database and a MySQL RDS database were launched to store time series values and metadata. A multi-tenancy environment was considered for database design, then time series metadata included information as the project and tenant the data belongs to, the S3 URL where the data resides, the file name, type and extension that identify the series, etc. The metadata database followed a shared-database-shared-schema mode, which allows tenants to manage their own metadata and business logic while ensuring database extensibility [Sarkar and Shah, 2015]. AWS Data Pipeline was used to export a DynamoDB table to a S3 bucket, to export data from a RDS table to a S3 bucket, and finally to copy data from a S3 bucket to another. For this purpose, a Data Pipeline template was downloaded from a public-access S3 repository containing AWS resources and modified in accordance to naming specifications for the specific AWS account. The architecture of this experiment is depicted on Fig. 6.11.

6.3.2 Deep Learning AMI

In order to test the deployment of a deep learning environment for the Batch Processing component, an AWS Deep Learning AMI was used to mount a TensorFlow framework in the AWS cloud as follows:

- An AWS Deep Learning AMI was unpacked to a C4.8xlarge-type EC2 instance, which is provisioned with 36 virtual CPUs, 60 GiB of memory, EBS-only storage, and 4,000 Mbps bandwith, as well as with multiple deep learning frameworks including TensorFlow.

- The TensorFlow framework was used to code a Python program that calculates non-linear predictions based on the Nearest Neighbors algorithm provided in [McClure, 2017]. This program receives as input the values for the embedding dimension and the delay between measurements required to map the time series into a collection of delay vectors. It also receives as input the number of neighgbors to be taken into account for calculating the non-linear prediction.

- A time series with 25,000+ values corresponding to the maximal daily temperature recorded by a CONAGUA's meteorological station in the

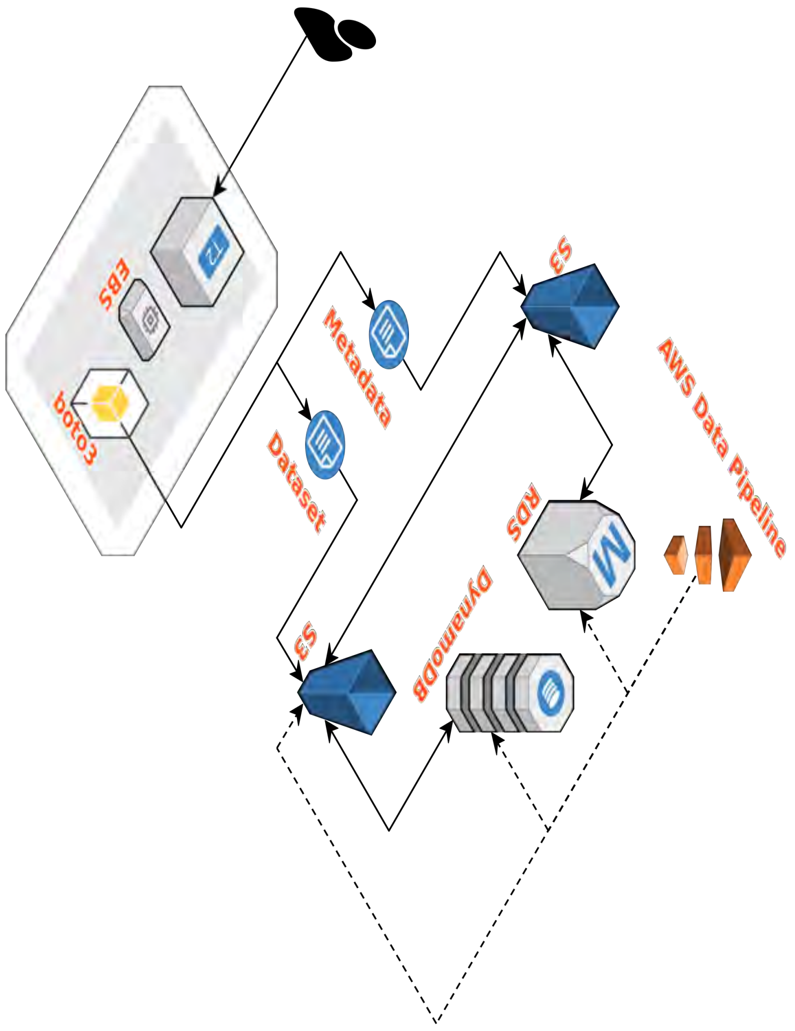

Figure 6.11: Architecture of the AWS experiment on storage, databases, and pipelines.

city of Morelia from 1947 to 2015, was serialized into the Pandas dataframe format, and then uploaded to a S3 bucket. The Python program was also uploaded to the same bucket.

- By using **boto3**, both the Python application and the serialized time series were copied from the S3 bucket to the EBS linked to the EC2 instance.
- The Python application was run on the EC2 instance. It removed lectures on the time series with missing values or outliers, and mapped the remaining values to a collection of 24,400+ delay vectors, which were randomly separated into an 80-percent training set and a 20-percent test set.
- Specific data types and operators from TensorFlow were used to exactly calculate the non-linear predictions based on the aforementioned Nearest Neighbors algorithm. This algorithm uses the *reduce-sum* operation as the basis for a broadcasting operation over arrays of diferent shape that speeds up calculations.

The architecture for this experiment is depicted in Fig. 6.12. It is important to bear in mind that the objective of this experiment was only to test the deployment of an AWS Deep Learning AMI for leveraging TensorFlow in the Batch Processing component of ITSFCC, and that a proper forecasting experiment on this platform requires an extensive configuration of the infrastructure and a solid planning and evaluation of the computing procedures, which are beyond the scope of this work. For this reason no further analysis on the data or results is provided.

6.3.3 Spark on Amazon EMR

In order to test the deployment of an Apache Spark environment for the Parallel Implementation component, a Spark cluster was launched in the Amazon EMR service. The experiment was implemented as follows:

- An Amazon EMR Spark cluster was launched on three M3.xlarge-type EC2 instances (1 master and 2 core nodes), which are provisioned with 4 virtual CPUs and 15 GiB of memory, as well as with Spark 2.3.0 on Hadoop 2.8.3.

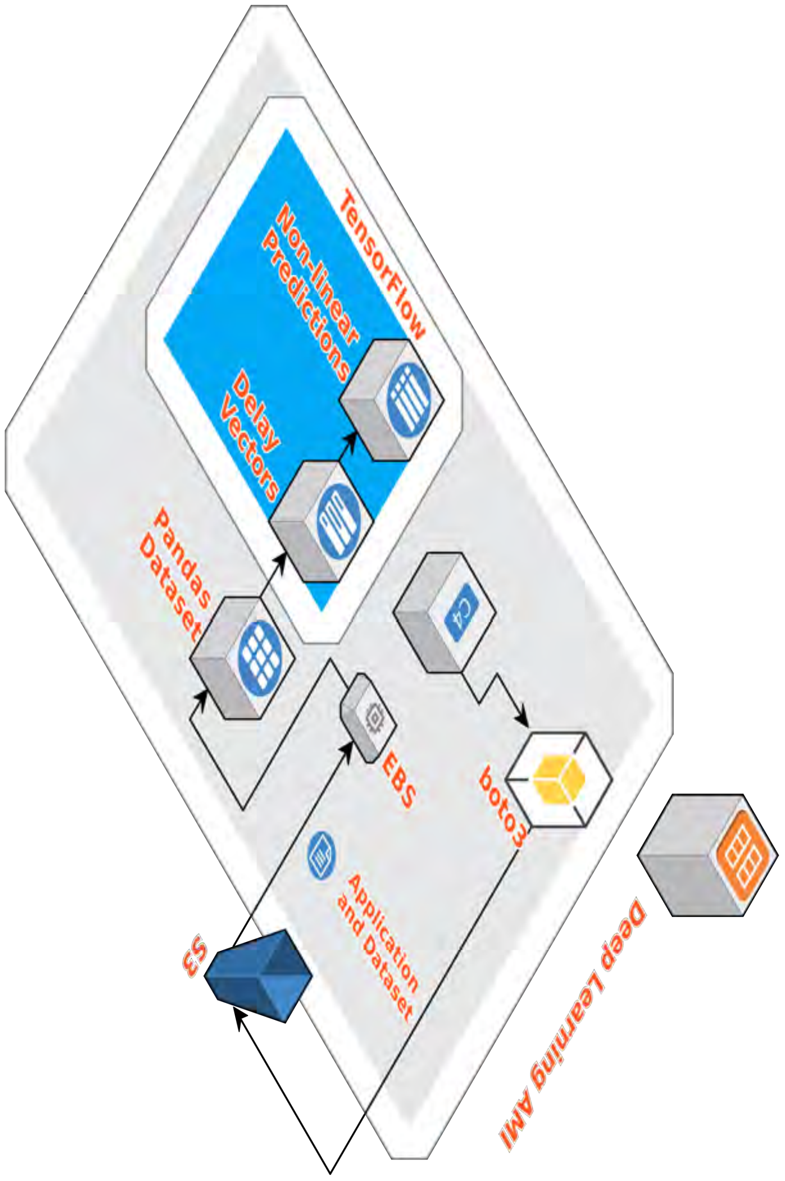

Figure 6.12: Architecture of the AWS experiment on the Deep Learning AMI.

- The Spark distributed processing system was used to code a set of Python snippets via the Spark Python application programming interface PySpark. These snippets were designed to apply basic filters to a time series, to run customized SQL queries against it, and to build interactive plots based on the time series via Bokeh.

- A time series with 5000+ daily values of the water level of a dam used for electric power generation, ranging from 2002 to 2017, was serialized into the Pandas dataframe format, and then uploaded to a S3 bucket.

- A PySpark snippet was run to remove lectures with missing values from the time series, then another snippet was executed to load the resulting time series into a Spark Data Frame by using a customized schema.

- A PySpark snippet was run to execute highly expressive SQL-based queries against the Data Frame using both SparkSQL queries, and Data Frame-based SQL-style functions. A complete description of these two methods of querying Apache Spark Data Frames can be found in [Kane, 2017] and in [Drabas and Lee, 2017].

- A PySpark snippet was run to produce a statistical description of the water level values in the Data Frame, and to produce a Bokeh interactive plot with the time series values.

The architecture for this experiment is depicted in Fig. 6.13. It is important to bear in mind that the objective of this experiment was only to test the deployment of a Spark environment for leveraging it in the Parallel Implementation component of ITSFCC, and that a proper forecasting experiment on this platform requires an extensive configuration of the infrastructure and a solid planning and evaluation of the computing procedures, which are beyond the scope of this work. For this reason no further analysis on the data or results is provided.

6.3.4 Virtual Private Cloud

In order to test the resources and services required to implement the Firewall Protection component, a sample virtual private cloud was launched by executing the following steps on Amazon VPC:

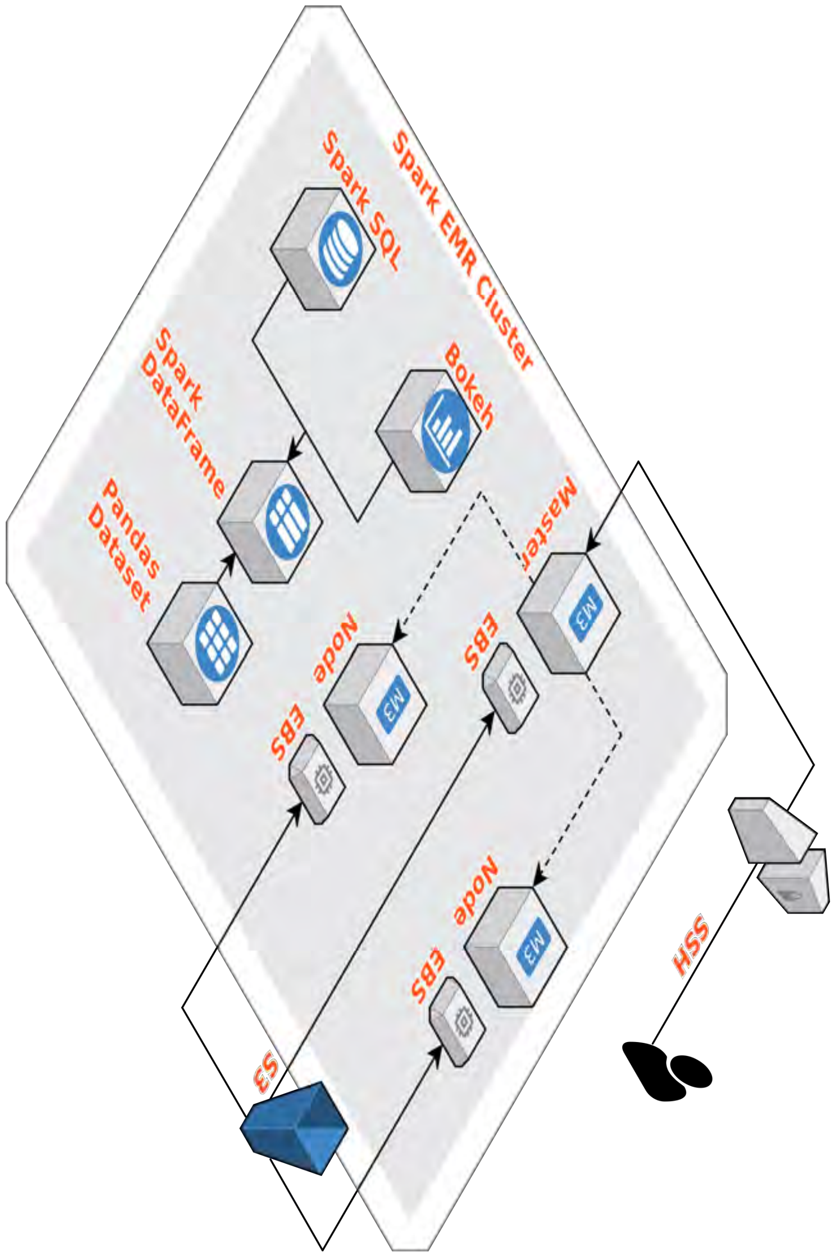

Figure 6.13: Architecture of the AWS experiment on the Spark EMR cluster.

- A virtual private cloud was started and then identified with the private IP address 10.0.0.0/16. Inside the VPC, a single EC2 instance was launched on a public subnet identified with the private IP address 10.0.1.0/24 (the interface subnet).
- By default, Amazon VPC provides an internet gateway for the VPC and places it in the public subnet, therefore the EC2 instance can connect to the internet via this gateway.
- The EC2 instance was assigned to a security group that allows inbound traffic over HTTP and HTTPS (ports 80 and 443) only, therefore it is open to REST-based requests and can be used as a web server.
- A second EC2 instance was launched on a private subnet, identified with the private IP address 10.0.4.0/24 (the processing subnet). This EC2 instance was assigned to a security group that allows inbound traffic from the interface subnet only, therefore it is hidden to the internet but open to the web server and can be used as an application server.
- The security group for the application server was modified to allow inbound traffic over SSH (port 22) from specific local IPs, therefore it is open to developers for maintenance purposes.
- A RDS database was launched in a private subnet identified with the private IP address 10.0.7.0/24 (the database subnet). This RDS instance was assigned to a security group that allows inbound traffic over MySQL (port 3306) from the processing subnet only, therefore the database is open for CRUD operations (Create, Read, Update, Delete) to the application server but hidden to users or applications in any other instance.
- The EC2 instance in the public subnet was given an elastic IP address, a static IPv4 address reachable from the Internet, so the web server can be exposed under a global identifer. Amazon Route53 can be used to translate this elastic IP address into a human-readable domain name like www.itsfcc.org.
- A NAT gateway was linked to the processing subnet, therefore the application server is able to communicate to the Internet to download software updates while remaining hidden to outside access.

The architecture for this experiment is depicted in Fig. 6.14. Even though this is a simple VPC configuration, its deployment can be time consuming and prone to human error. For this reason it is advisable to compile all of the aforementioned steps as a template for AWS CloudFormation. This is a service that allows to programatically deploy AWS stacks based on YAML template files, either on the AWS Management Console or the AWS Command Line Interface, in compliance with an approach known as Infrastructure-as-Code. Examples of typical Infrastructure-as-Code operations for networking in the AWS cloud can be found in [Das and Modi, 2017].

6.3.5 Bokeh Server Customized AMI

In order to test the deployment of an interactive visualization environment for the Engagement Layer component, a Bokeh server was configured on the AWS cloud as follows:

- An EC2 M5.xlarge-type instance was launched on AWS. This instance was provisioned with 4 virtual CPU's, 16 GiB of memory, and 8 GiB of EBS, as well as with Ubuntu and Anaconda. A Bokeh server was installed on the EC2 instance. Bokeh is an interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of versatile graphics, and to extend this capability with high-performance interactivity over very large or streaming datasets [Anaconda, 2015].

- A set of time series containing measures of the quality of electric power in five sub-station circuits, recorded each 10 minutes from January 2015 to November 2017 (140,000+ lectures), were uploaded to a S3 bucket. The time series were previously used to produce 7-day forecasts, starting on each one of the last 28 days of the original period, for each circuit. As a result, a total of 5x28 time series comprised of 7x144 forecast values were obtained. These forecast values were also uploaded to the S3 bucket.

- A Python program was coded to request services from the Bokeh server. This program leveraged **boto3** to download from S3 the original data, as well as the 5x28 time series that resulted from forecasting. Time series were translated from CSV to the Pandas dataframe format, and then passed to Bokeh methods to build the interactive plots.

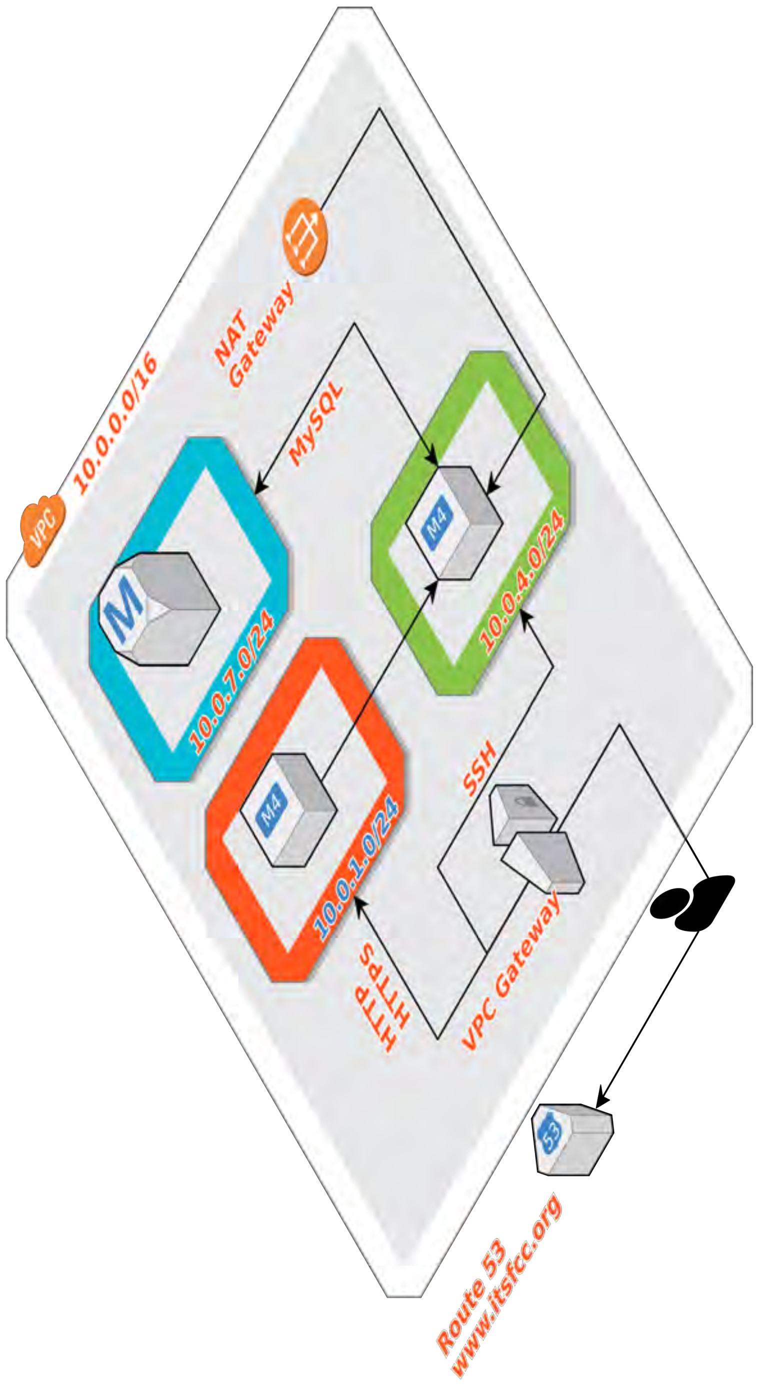

Figure 6.14: Architecture of the AWS experiment on the virtual private cloud.

- A Graphic User Interface (GUI) was implemented to allow a Business Intelligence (BI) user to visualize a specific forecast period and a summary of results by selecting a circuit and a starting date. The GUI also allowed the BI user to send SMS-based alarms to specific stakeholders if the predicted energy quality measures dropped below pre-defined thresholds. For this purpose, an Amazon SNS Pub/Sub messaging channel was setup with specific notification topics, a list of subscribers, and their phone numbers.

- The EC2 instance was given an Elastic IP address to make the interactive visualization available to remote computers authorized by the firewall defined via Amazon IAM security groups.

- Finally, a customized Amazon Machine Image was built with the fully deployed Bokeh server, including the EC2 instance configuration and the contents in the EBS volume. By launching this customized AMI, the time required for complete re-deployment of the Bokeh server dropped from 20+ to 2 minutes.

The architecture for this experiment is depicted in Fig. 6.15. It is important to bear in mind that the objective of this experiment was only to test the deployment of a Bokeh-based environment for the Engagement Layer component of ITSFCC, and that a proper user interface development on this platform requires an extensive configuration of the infrastructure as well as a complete design of interaction strategies, which are beyond the scope of this work. For this reason no further information regarding the data or the user interface is provided.

6.4 Results

This section summarizes the results achieved by this thesis in relation to the objectives stated in Section 1.3, as follows:

- An architecture characterization, based on a partial implementation of the ISO/IEC/IEEE 42010 standard for architecture description, was designed to express the ITSFCC architecture. This architecture characterization includes basic documentation concerning the system, its stakeholders, and the stakeholder's concerns the system addresses. The architecture characterization also includes an architecture model which

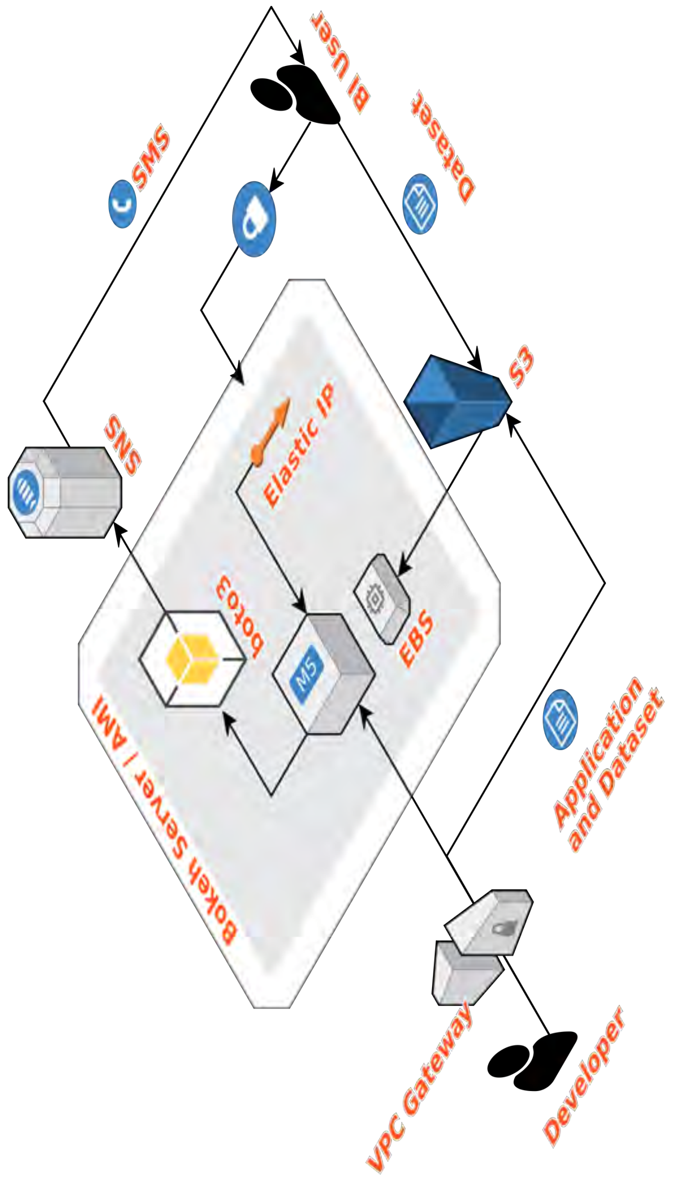

Figure 6.15: Architecture of the AWS experiment on the Bokeh server customized AMI.

is oriented to a process view, expressed via block diagrams, and based on software patterns. As long as it is a partial implementation of the standard, this architecture characterization can be eventually expanded in order to fully comply with the overall specifications of an architecture description.

- Based on the aforementioned architecture characterization, an abstract architecture model for ITSFCC was built. This cloud-native, pattern-based, provider-independent architecture comprises 15 application components designed to provide a set of features the system requires to address the problem of intelligent time series forecasting and to leverage the potential of the cloud computing paradigm. The application components are also designed to meet the software quality attributes required by the system's stakeholders under normal operation.

- A reference architecture for deploying ITSFCC on the AWS cloud was built. This reference architecture comprises 9 of the 15 application components in the abstract architecture, as long as the other components are already included in AWS's core functionality. For each one of these 9 application components, a block diagram expressed as a Cloudcraft blueprint, as well as a guide for deploying the corresponding infrastructures and platforms on the AWS cloud are provided. This reference architecture can be considered as a pioneering work, as long as no other effort to integrate time series forecasting, Artificial Intelligence and Machine Learning-based methods of forecasting, and cloud computing was found in the literature.

Summary

Based on the abstract architecture characterization for ITSFCC proposed in Chapter 5, this chapter presented a concrete, provider-specific architecture for the Amazon Web Services cloud. An overview of the specific services required to implement ITSFCC components on AWS was provided. A corresponding set of implementation guides, consisting of Cloudcraft blueprints directed, via numbered pointers, to detailed explanations was also presented. Finally, a set of experiments deployed on AWS in order to validate the adequacy of specific services to the requirements placed by ITSFCC components was summarized. Among these experiments, several concrete applications of cutting edge platforms on the AWS cloud stood out as promising subjects for future research. That is the case of the deep learning framework Tensor-Flow, the distributed processing system Apache Spark, and the interactive visualization library Bokeh. Next chapter presents the general conclusions of this work, as well as a set of directives for future work.

Chapter 7

Conclusions and Future Work

This chapter presents the conclusions of the thesis, as well as set of directions for future work. A summary with the main points covered in the thesis is given as the general conclusion. Specific conclusions regarding the contributions made by the thesis and recommendations for extending this work are also given. Finally, directives for future work on ITSFCC are grouped into two categories: actions concerning FaaS operation, and actions related to expanding it into a cloud-based platform for collaborative research.

7.1 Conclusions

7.1.1 General Conclusion

A software architecture was designed as the first step towards the development of the Intelligent Time Series Forecasting based on Cloud Computing system. The model of a cloud-native application architecture was selected to enable the system to provide a Forecasting-as-a-Service operation, as well as to setup the basis for a collaborative research platform. The architecture was designed following a top-down approach: it started with analyses of the system context in terms of the problems inherent to intelligent time series forecasting and the resources cloud computing offers to address such problems. Context description was mapped to broad features the system is required to exhibit, then features were translated into application components, and finally to precise software quality attributes. The architecture was designed as an abstract, provider-independent product, able to be transferred to different cloud offerings and deployment models. Nevertheless, in order to validate its adequacy to a concrete situation, it was also translated to a provider-specific architecture for the Amazon Web Services cloud. AWS was selected because of the extension, maturity, and market leadership of its services. Software patterns for cloud offering components, cloud application architecture, cloud application management, and Big Data and advanced analytics provided a basis for the design of ITSFCC architecture.

7.1.2 Contributions

Following are the contributions made by this thesis:

- The thesis produced original content by addressing a very specific, contemporary research field. As a matter of fact, no other project working on the design of a cloud-native application architecture for intelligent time series forecasting was found when searching for architecture, model, or implementation references.

- In order to deal with the lack of reference architectures in the very specific field this thesis addressed, multiple information resources were combined into a systematic collection that included results from formal research on software and systems architecture, software patterns, cloud computing in relation to computer systems, cloud computing in relation to service sciences, Big Data analytics, energy time series analysis and forecasting, Artificial Intelligence and Machine Learning-based forecasting tools, and also concrete reference architectures produced by cloud providers. This hybridization of information resources can be considered as part of an interdisciplinary approach required to deal with contemporary technology-based projects that exhibit high complexity.

- Unlike most of the research projects working on time series forecasting or intelligent time series forecasting, which emphasize in results presented as models or algorithms, this thesis focused on architectures, patterns, and data engineering as the backbone that internally supports the model/algorithm deployment. This approach is driven by the fact that, when it comes to operating at production stage, forecasting is required to be not only accurate, but also efficient in terms of speed and performance. In this context, leveraging cloud computing for fast results over complex processing structures and huge datasets requires to simultaneously think about the algorithm and the architecture, in compliance with the interdisciplinary approach mentioned in the previous item.

- The abstract architecture presented in this thesis can be regarded as a valuable link between the vast amount of formal research previously conducted on software and systems architecture and the also extensive catalog of informal reference architectures produced by cloud providers to address frequent use-scenarios. In this context, the elements suggested by this thesis for ITSFCC architecture characterization (process-based architecture view, block-diagram model kind, and software pattern-based configuration) provide a basis for a formal study on the huge collection of empiric architectures built for the AWS cloud so far.

- As a result of analysing the ITSFCC context in terms of the problems inherent to intelligent time series forecasting (Chapter 3) this thesis suggested that hybridization is worth to be studied as a productive strategy for building Artificial Intelligence and Machine Learning-based methods of forecasting. Research in this direction can be started by analysing hybrid methods studied by the Work Group, in addition to the Nearest Neighbors Differential Evolution algorithm, and by summarizing other promising hybrid tools to be explored in the future.

7.1.3 Recommendations

Following are recommendations the author of this thesis considers valuable for researchers interested in working on parallel projects:

- The conceptual framework this thesis presented around software and systems architecture (Chapter 2) can be expanded by including a section dealing with the concept of *enterprise architecture*, as a high-level view of an organization's information-related components that conveys an overall understanding of each component and an understanding of the relationship and the interaction between these components [Mahmood and Hill, 2011]. Enterprise architectures encompass software architectures as part of a higher-level organizing principle that aligns a functional business mission with the IT strategy and execution plans. Descriptions and application examples of popular enterprise architecture frameworks like Zachman's or The Open Group Architecture Framework (TOGAF) [Raj et al., 2017] are also recommended to be included.

- The technological and organizational prerequisites for cloud computing adoption discussed in Chapter 4 can be complemented with a set of guidelines to more specific changes that are also required. Those changes are grouped in [Stine, 2015] with the overall theme of decentralization and autonomy. Among the most relevant of those changes are: decentralization of skill sets into cross-functional teams (DevOps), decentralization of the release schedule and process (continuous delivery), control of application packaging distributed to business capability teams (containerization), and control of individual business capabilities distributed to individual autonomous services (microservices).

- A preliminar analysis of the requirements and implications of deploying ITSFCC on cloud offerings and deployment models alternative to AWS is highly recommended. This analysis should include AWS' contenders in the public-cloud market, like Microsoft Azure, Google Cloud Platform, or Digital Ocean. The analysis can also be extended to deploying ITSFCC on a private, community, or hybrid cloud instead of on a public one.

7.2 Future Work

7.2.1 ITSFCC FaaS Operation

Future work on ITSFCC concerning its FaaS operation include the following actions:

- To develop, on the basis of the abstract ITSFCC architecture characterization, a complete plan for implementing the system using NNDE as an initial and representative model of forecasting, as it was used for architecture design. This plan will have to include a timeline for designing, building, and deploying each application component, as well as a set of guidelines to test individual and overall functionality. Once this plan is completed, the implementation of ITSFCC can be properly scheduled as part of the Work Group's production.

- To enhance the ITSFCC functionality by merging the Batch Processing and Parallel Implementation components, to perform parallel processing to the greatest possible extent. Eventually, this action will result in the deployment of just one application component able to process data in a serial or parallel way, depending on the characteristics of time series operations or parameter optimization to be performed.

- To enhance the functionality of the Graph Representation component by storing time series operations in true graph databases instead of in key-value storage, as it is currently designed. As long as this application component uses graphs to manage complex operation sequences, it can be extended to provide an abstraction layer for the processing structures used by advanced frameworks such as Spark or TensorFlow, which are graph-based as well.

- To design and develop an experimental application component able to process a continual stream of data coming from, for instance, Internet of Things devices reading energy-related variables at a high-rate. The development and testing of this component can provide valuable insights for future streaming operation in specific data flows of the system. The Spark infrastructure in the Parallel Implementation component can be used as a starting point for stream processing, however an analysis of moving this functionality to a Flink environment is highly advised.

7.2.2 ITSFCC as a Platform for Collaborative Research

Future work in relation to expanding ITSFCC into a cloud-based platform for collaborative research include the following actions:

- To develop a complete plan to convert the ITSFCC architecture into a unified environment for collaboration in data science projects, starting with the researchers and graduate students affiliated to the Work Group. Key issues to address in this plan are the definition of standards for data management and security, as well as for sharing research results.

- Regarding the standardization of data management, full specifications for shared data repositories, data extraction and transformation operations, and processing workflows, via the Graph Representation component, will have to be designed.

- Regarding data security, an extensive set of internal regulations for ensuring confidentiality, integrity and availability of data will have to be produced.

- Regarding the process of sharing research results, an exhaustive procedure for storing fully detailed experiments as shared data, machine images, or workflows will have to be prepared. This procedure will also have to provide precise instructions for a different researcher to remotely recover the experiments from shared resources.

The importance of future work on ITSFCC as a platform for collaborative research cannot be overstated. Cloud-based collaboration allows academic institutions to enhance research workflows across multiple laboratories or teams, as well as to speed up the generation, variation, and interpretation of experiments from different viewpoints. As a consequence, high-complexity research fields, such as intelligent time series forecasting, can be addressed by interdisciplinary groups, able to examine the problems from diverse and complementary perspectives, and to collectively propose an enriched approach to the solution. In this context, cloud computing must be regarded not only as a way to improve the efficiency of current research procedures, but also as a foundation for the development of future, interdisciplinary research strategies.

References

T Afanasieva, A Sapunkov, and A Afanasiev. Software of time series forecasting based on combinations of fuzzy and statistical models. 2018.

Marco Aiello, Einar Broch Johnsen, Schahram Dustdar, and Ilche Georgievski. *Service-Oriented and Cloud Computing*. Springer, 2016.

Anaconda. Welcome to Bokeh. `http://bokeh.pydata.org`, 2015. [Online; accessed: 2018-04-02].

AWS. Amazon Web Services. `http://aws.amazon.com`, 2018. [Online; accessed: 2018-04-02].

Christian Baun, Marcel Kunze, Jens Nimis, and Stefan Tai. *Cloud Computing: Web-Based Dynamic IT Services*. Springer Science & Business Media, 2011.

Enrique Castro-Leon and Robert Harmon. *Cloud as a Service: Understanding the Service Innovation Ecosystem*. Apress, 2016.

Lucas Chan and Rowan Udell. *AWS Administration Cookbook*. Packt Publishing Ltd, 2017.

Cloudcraft. Cloudcraft. `http://cloudcraft.co`, 2018. [Online; accessed: 2018-04-02].

Melvin E Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

Houda Daki, Asmaa El Hannani, Abdelhak Aqqal, Abdelfattah Haidine, and Aziz Dahbi. Big data management in smart grid: concepts, requirements and implementation. *Journal of Big Data*, 4(1):13, 2017.

L. Dannecker. *Energy Time Series Forecasting: Efficient and Accurate Forecasting of Evolving Time Series from the Energy Domain.* Springer Fachmedien Wiesbaden, 2015.

Satyajit Das and Jhalak Modi. *AWS Networking Cookbook.* Packt Publishing Ltd, 2017.

Shyam Kumar Doddavula, Ira Agrawal, and Vikas Saxena. Cloud computing solution patterns: Application and platform solutions. In *Cloud Computing*, pages 221–239. Springer, 2013a.

Shyam Kumar Doddavula, Ira Agrawal, and Vikas Saxena. Cloud computing solution patterns: Infrastructural solutions. In *Cloud Computing*, pages 197–219. Springer, 2013b.

Tomasz Drabas and Denny Lee. *Learning PySpark: build data-intensive applications locally and deploy at scale using the combined powers of Python and Spark 2.0.* Packt Publishing Ltd, 2017.

Schahram Dustdar, Frank Leymann, and Massimo Villari. *Service-Oriented and Cloud Computing.* Springer, 2015.

Morris J Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. Technical report, 2007.

Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications.* Springer Science & Business Media, 2014.

Juan J Flores, Roberto Loaeza, Héctor Rodríguez, and Erasmo Cadenas. Wind speed forecasting using a hybrid neural-evolutive approach. In *Mexican International Conference on Artificial Intelligence*, pages 600–609. Springer, 2009.

Juan J Flores, José R Cedeño González, Rodrigo Lopez Farias, and Felix Calderon. Evolving nearest neighbor time series forecasters. *Soft Computing*, pages 1–10, 2017.

Ian Gorton. *Essential software architecture.* Springer Science & Business Media, 2006.

Mohammed Guller. *Big data analytics with Spark: A practitioner's guide to using Spark for large scale data analysis*. Springer, 2015.

Markus Helfert, Víctor Méndez Muñoz, and Donald Ferguson. *Cloud Computing and Services Science*. Springer, 2016.

Markus Helfert, Donald Ferguson, Víctor Méndez Muñoz, and Jorge Cardoso. *Cloud Computing and Services Science*. Springer, 2017.

Wei-Chiang Hong. *Intelligent energy demand forecasting*. Springer, 2013.

ISO/IEC. *ISO/IEC 19501: information technology - open distributed processing - unified modeling language (UML) version 1.4.2*. ISO, 2005. URL `https://books.google.com.mx/books?id=2k-6kQEACAAJ`.

ISO/IEC/IEEE. ISO/IEC/IEEE Systems and software engineering – Architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, Dec 2011. doi: 10.1109/IEEESTD.2011.6129467.

Frank Kane. *Frank Kane's Taming big data with Apache Spark and Python: real-world examples to help you analyze large datasets with Apache Spark*. Packt Publishing Ltd, 2017.

Veselin Kantsev. *Implementing DevOps on AWS*. Packt Publishing Ltd, 2017.

Holger Kantz and Thomas Schreiber. *Nonlinear time series analysis*, volume 7. Cambridge university press, 2004.

P. B. Kruchten. The 4+1 View Model of architecture. *IEEE Software*, 12 (6):42–50, Nov 1995. ISSN 0740-7459. doi: 10.1109/52.469759.

James Lewis and Martin Fowler. Microservices: a definition of this new architectural term. *MartinFowler. com*, 25, 2014.

Frank Leymann, Uwe Breitenbücher, Sebastian Wagner, and Johannes Wettinger. Native cloud applications: Why monolithic virtualization is not their foundation. In *International Conference on Cloud Computing and Services Science*, pages 16–40. Springer, 2016.

Rodrigo Lopez Farias, Vicenç Puig, Hector Rodriguez Rangel, and Juan J Flores. Multi-model prediction for demand forecast in water distribution networks. *Energies*, 11(3):660, 2018.

Zaigham Mahmood and Richard Hill. *Cloud Computing for enterprise architectures*. Springer Science & Business Media, 2011.

Jesus Maillo, Sergio Ramírez, Isaac Triguero, and Francisco Herrera. knn-is: An iterative Spark-based design of the k-Nearest Neighbors classifier for big data. *Knowledge-Based Systems*, 117:3–15, 2017.

Nick McClure. *TensorFlow Machine Learning Cookbook*. Packt Publishing Ltd, 2017.

Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. 2011.

Amit Nandi. *Spark for Python Developers*. Packt Publishing Ltd, 2015.

Jordan Novet. Amazon lost cloud market share to Microsoft in the fourth quarter: Keybanc, 2018. URL `http://www.cnbc.com/2018/01/12/amazon-lost-cloud-market-share-to-microsoft-in-the-fourth-quarter-keybanc.html`. [Online; accessed: 2018-03-30].

Anand Balachandran Pillai. *Software architecture with Python: design and architect highly scalable, robust, clean, and high performance applications in Python*. Packt Publishing Ltd, 2017.

Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

Pethuru Raj, Anupama Raman, and Harihara Subramanian. *Architectural Patterns: Uncover essential patterns in the most indispensable realm of enterprise architecture*. Packt Publishing Ltd, 2017.

Hector Rodriguez Rangel, Vicenç Puig, Rodrigo Lopez Farias, and Juan J Flores. Short-term demand forecast using a bank of neural network models trained using genetic algorithms for the optimal management of drinking water networks. *Journal of Hydroinformatics*, 19(1):1–16, 2017.

Hector Rodriguez, Vicenç Puig, Juan J Flores, and Rodrigo Lopez. Combined holt-winters and ga trained ann approach for sensor validation and reconstruction: Application to water demand flowmeters. In *Control and Fault-Tolerant Systems (SysTol), 2016 3rd Conference on*, pages 202–207. IEEE, 2016.

Aurobindo Sarkar and Amit Shah. *Learning AWS: design, build, and deploy responsive applications using AWS cloud components*. Packt Publishing Ltd, 2015.

Nitin Sawant and Himanshu Shah. *Big Data Application Architecture Q&A: A Problem-Solution Approach*. Apress, 2014.

Manish Sethi. *Cloud Native Python*. Packt Publishing Ltd, 2017.

R Shyam, Bharathi Ganesh HB, Sachin Kumar, Prabaharan Poornachandran, and KP Soman. Apache Spark: a big data analytics platform for smart grid. *Procedia Technology*, 21:171–178, 2015.

Matt Stine. Migrating to cloud-native application architectures, 2015.

Nassim Nicholas Taleb. *Antifragile: Things that gain from disorder*, volume 3. Random House Incorporated, 2012.

Kwa-Sur Tam and Rakesh Sehgal. A cloud computing framework for on-demand forecasting services. In *International Conference on Internet of Vehicles*, pages 357–366. Springer, 2014.

Diego Teijeiro, Xoán C. Pardo, Patricia González, Julio R. Banga, and Ramón Doallo. *Implementing Parallel Differential Evolution on Spark*, pages 75–90. Springer International Publishing, Cham, 2016. ISBN 978-3-319-31153-1. doi: 10.1007/978-3-319-31153-1_6. URL `https://doi.org/10.1007/978-3-319-31153-1_6`.

Ariel Tseitlin. The antifragile organization. *Communications of the ACM*, 56(8):40–44, 2013.

Jinesh Varia. Architecting for the cloud: Best practices. *Amazon Web Services*, 1:1–21, 2010.

Uchit Vyas. *Mastering AWS Development*. Packt Publishing Ltd, 2015.

Yohan Wadia. *AWS Administration–The Definitive Guide*. Packt Publishing Ltd, 2016.

Adam Wiggins. The twelve-factor app. *factor. net*, 2014.

Raghu Yeluri, Enrique Castro-Leon, Robert R Harmon, and James Greene. Building trust and compliance in the cloud for services. In *SRII Global Conference (SRII), 2012 Annual*, pages 379–390. IEEE, 2012.

Giancarlo Zaccone. *Getting Started with TensorFlow*. Packt Publishing Ltd, 2016.