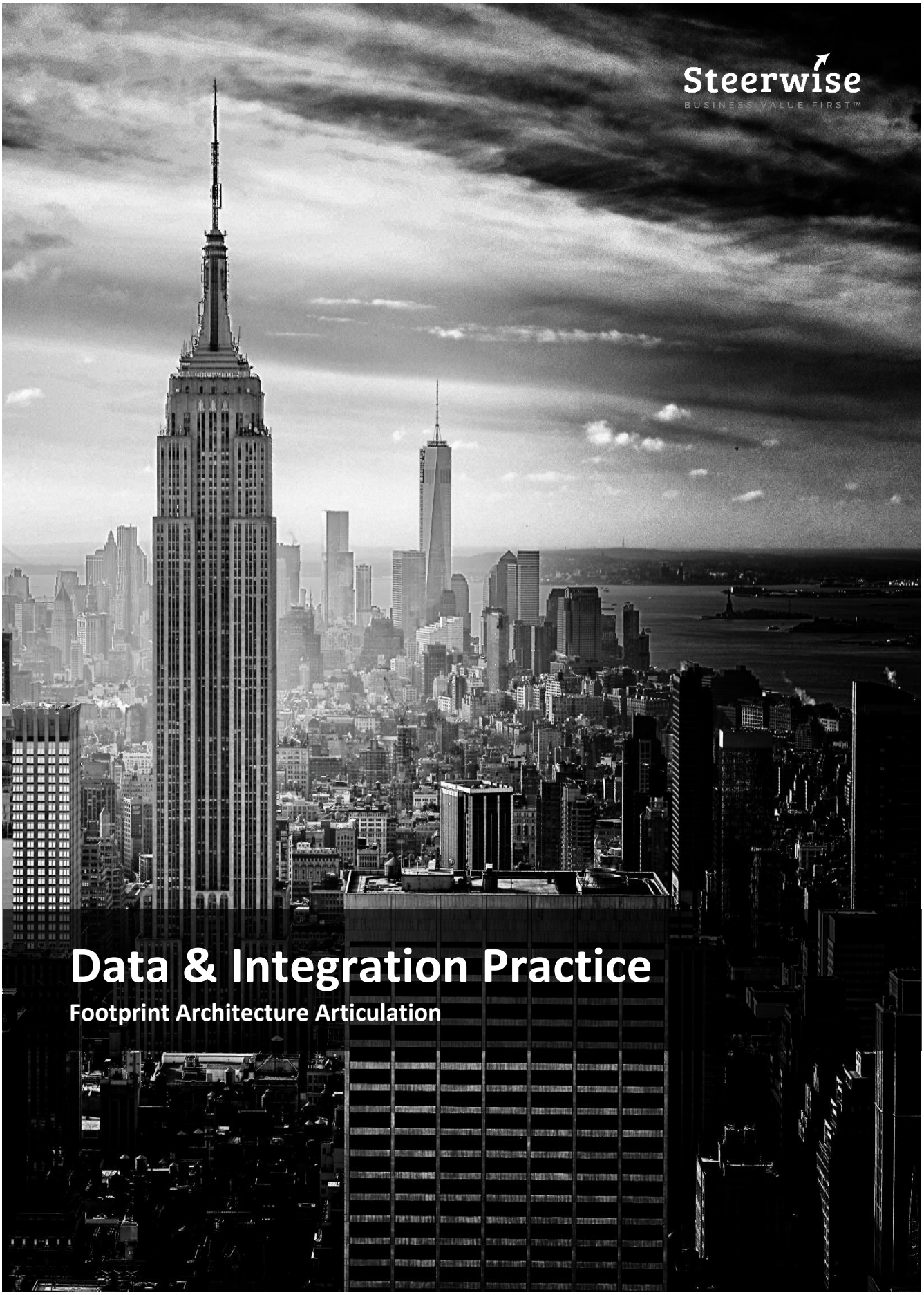
Steerwise
BUSINESS VALUE FIRST™
Data & Integration Practice
Footprint Architecture Articulation

Enterprise Data Platform

Footprint Architecture

Table of Contents

1.0 Enterprise Data Platform Footprint Architecture

1.1 EDP Footprint Architecture Overview

To successfully implement the EDP Blueprint Architecture, a footprint of cloud-native based digital technology architecture has been defined. Footprint Architecture depicts the deployment of key vendor technology platforms & tools at the **infrastructure & platform** levels of the stack.

Infrastructure & Platform implementation is aligned with Cloud Native Computing Foundation (***CNCF***) based architecture. Higher reliability and availability of infrastructure & Platforms are achieved by deploying on ***Cloud Platform*** for scalable orchestration & easy maintenance. The infrastructure & platform architecture enables applications to be rolled out through ***Continuous Integration*** & ***Continuous Delivery*** on ***containerized*** environment. ***Shift left*** practice is adopted to incorporate **security** at all stages starting from the beginning.

The **Footprint Architecture** leverages existing Vendor Technology Platforms & Tools at TFS where appropriate and introduces advanced Vendor Technology Platforms & Tools where necessary to fulfill the prioritized requirements.

The **Footprint Architecture** will continue to evolve with additional set of Vendor Technology Platforms & Tools based on the ongoing & future requirements.

1.2 EDP Infrastructure Footprint Architecture

The **EDP Blueprint Architecture** for **Infrastructure Services** is realized through the following **Footprint Architecture** that is designed based on the initial set of requirements for EDP.

The **EDP Infrastructure Footprint Architecture** is depicted below:

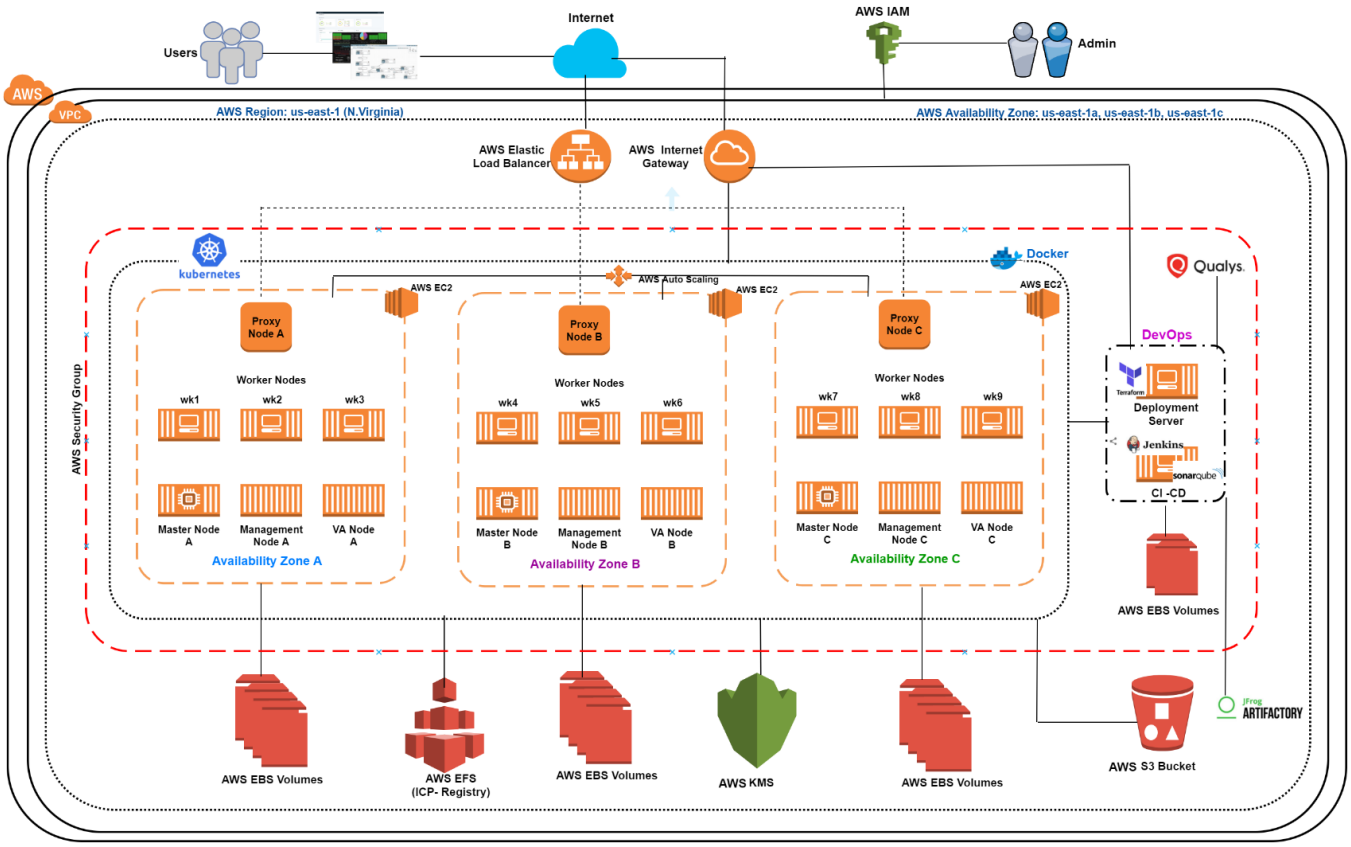

Fig 1: *EDP Infrastructure Footprint Architecture*

The **Infrastructure Footprint Architecture** provides the following capabilities:

- Deployment is carried out from an AWS EC2 instance using ***Terraform***.
- Infrastructure deployed on ***AWS Public Cloud*** within us-east-1 region.
- TFS provided ***VPC***, ***Subnet*** & ***Security Groups*** used for deployment.
- High Availability achieved by deploying the infrastructure in *three* ***Availability Zones***.
- ***AWS EC2*** instances provisioned to host the Applications and Database.
- ***AWS Elastic Block Storage*** used for Persistent Data Store.
- ***AWS S3*** used as Object Store.
- ***AWS Elastic Load Balancer*** used for accessing Management Console.

- ***AWS Bastion Host*** deployed in public subnet to protect applications on private subnet.
- ***Kubernetes*** Cluster deployed using customized ***AMI***.
- Two additional nodes are deployed for *Management* and *Vulnerability Advisor*.
- ***AWS IAM*** Policy & Roles are used for securing the resources.
- ***AWS KMS*** is used to store the security keys.
- ***AWS Auto Scaling*** is recommended for ***Elasticity*** of EC2 resources.
- An EC2 instance is deployed outside of Kubernetes Cluster for ***DevOps*** to implement ***Continuous Integration*** & ***Continuous Delivery (CI-CD).***
- Leverages ***Qualys*** to perform ***Vulnerability*** scan of container images before deployment.
- Integrate ***Artifactory*** to store internal & external artifacts.
- ***SonarQube*** integrated with ***CI-CD*** process to carry out ***code quality*** & ***security scan***.

1.3 EDP Platform Footprint Architecture

The **EDP Blueprint Architecture** for **Platform Services** is realized through the following **Footprint Architecture** that is designed based on the initial set of requirements for EDP.

The **EDP Platform Footprint Architecture** is depicted below:

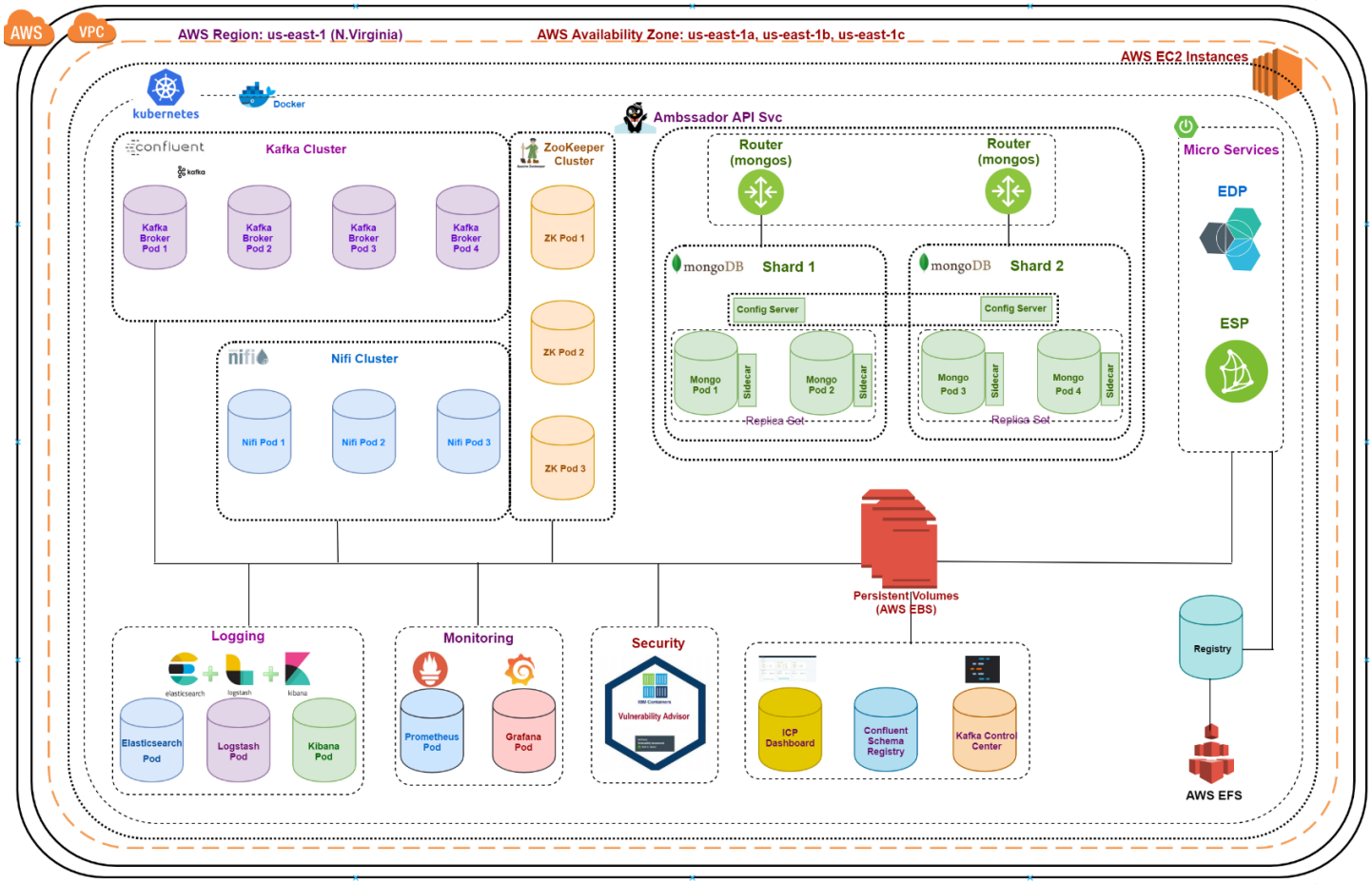

Fig 2: ***EDP Platform Footprint Architecture***

Technology platform components encompasses state of art technologies for achieving high reliability and performance. ***Cloud Native*** components blended with best ***Security*** practice implemented across all parts of the implementation. Operation efforts are made easy by implementing end to end ***Logging*** and ***Monitoring***.

1.3.1 Data Processing Platform

1.3.1.1 Kafka Cluster

Kafka takes a key role in data access & delivery services. Kafka scales horizontally and fully fault tolerant. Kafka cluster stores streams of *records* in categories called *topics* & each record consists of a key, a value & a timestamp.

- Kafka Brokers are deployed as pods on Kubernetes.
- Four Kafka Broker pods deployed on AWS EC2 instances running on three AZs.
- Kafka Broker pods run in cluster mode orchestrated by ZooKeeper.
- Kafka Broker pods deployed as StatefulSets with Anti-Affinity.
- Each Broker pod is tied to EBS persistent volume.
- Configuration handled via ConfigMap.
- Producers and Consumer topics handle the ingress data.

1.3.1.2 NiFi Cluster

NiFi helps to automate flow of data between systems in asynchronous mode by allowing high throughput with natural buffering even as processing and flow rates fluctuate. NiFi *Flow Controller* provides threads for extensions to run on & manages the schedule of when extensions receive resources to execute. NiFi employs the model of *Zero-Master Clustering* where by each node in a NiFi cluster performs the same tasks on the data, but each operates on a different set of data.

- NiFi Nodes are deployed as pods on Kubernetes.
- Three NiFi pods deployed on AWS EC2 instances running on three Availability Zones.
- NiFi pods run in cluster mode orchestrated by ZooKeeper.
- NiFi pods deployed as StatefulSets with Anti-Affinity.
- Each pod is tied to EBS persistent volume.
- Configuration handled via ConfigMap.

1.3.1.3 Zookeeper Cluster

ZooKeeper is used as coordination service for distributed applications to implement higher level services for synchronization, configuration maintenance, and groups and naming. It helps to relieve distributed applications the responsibility of implementing coordination services from scratch. Zookeeper maintains an in-memory image of state, along with transaction logs and snapshots in a persistent store.

- Zookeeper Nodes are deployed as pods on Kubernetes.
- Three Zookeeper pods deployed on AWS EC2 instances running on three Availability Zones.
- Zookeeper pods deployed as StatefulSets with Anti-Affinity.
- Each pod is tied to EBS persistent volume.
- Configuration handled via ConfigMap.
- Zookeeper helps in coordination of the service with Kafaka and NiFi applications.

1.3.2 Data Storage Platform

1.3.2.1 Object Store

AWS S3 used as ***Object Store*** to acquire or receive data from data sources and persist ***Raw Data*** objects in native format. AWS S3 is also used as ***Egression Data Store.*** Data in S3 is ***encrypted*** at rest.

1.3.2.2 Harmonized & Materialized Data Store

MongoDB document database provides high performance data persistence, high availability, and automatic scaling. Indexes support faster queries and can include keys from embedded documents and arrays. MongoDB supports horizontal scaling through ***sharding***. Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set. The ***mongos*** act as a ***query router***, providing an interface between client applications and the sharded cluster. ***Config servers*** store metadata and configuration settings for the cluster. The metadata reflects state and organization for all data and components within the sharded cluster. The metadata includes the list of chunks on every shard and the ranges that define the chunks.

- MongoDB Nodes are deployed as pods on Kubernetes.
- Four MongoDB pods deployed on AWS EC2 instances running on three Availability Zones.
- MongoDB pods deployed in two shards cluster with two pods each.
- Shards deployed as Stateful Replica Sets on to dedicated nodes with anti-affinity.
- Two instances of Config Servers are deployed as a replica set.
- Pods use Mongo Sidecar for automatic Replica Sets configuration.
- Persistent EBS volumes used for data store.
- Data is encrypted both in-transit and at-rest and makes it easy to control access with role-based user management.