Automated Testing of Data Survivability and Restorability

Sylvain Muller and Ciarán Bryce

University of Applied Sciences (HES-SO),
Geneva, Switzerland

Abstract. Regular data backups are fundamental for protection against cyber-attacks and damage to infrastructure. To ensure a successful restoration, backed up data must be tested regularly for restorability to the company's current environment. Cloud providers generally test their backed-up data, but a testing framework is also required for locally stored files and databases. The paper proposes an automated test framework that validates the continued usability of backed up data for target restoration environments. The framework tests backups of Excel files, MySQL and Postgres databases, PDF documents and flat files.

Keywords Security, backup, automation, testing, infrastructure-as-code.

1 Introduction

Data is a company's most valuable asset. However, cyber-security attacks like ransomware or physical events like damage, loss or theft are serious threats to data in all companies. Effecting data backups is fundamental to companies' security and survivability since this is the only way company data can survive a physical or cyber-emergency [4].

Many companies have a regrettable tendency to forget that backup is only half of the matter. Restoration must be an efficient and well practiced process since, contrary to backups, restorations are initiated in emergency conditions and failure to restore correctly undoes the benefit of doing the backup in the first place.

Two conditions must be met for a data restoration to succeed. First, the backed up data, or *snapshot*, must survive until the restoration, and be readable during the operation. Second, the environment in which the snapshot is restored must be able to accommodate the restored data. The data may be unreadable in a different environment to the one in which the snapshot is made due to incorrect software versions. This can happen if a company moves location after an emergency or if there is a significant change in the environment between backup and restoration.

Companies have been moving to the cloud over the past decade, partly to delegate data backup and restoration issues. However, not all SMEs trust or are ready to move to the cloud. They prefer to manage data locally, and consequently, require a means to ensure backup and restoration for their files and databases.

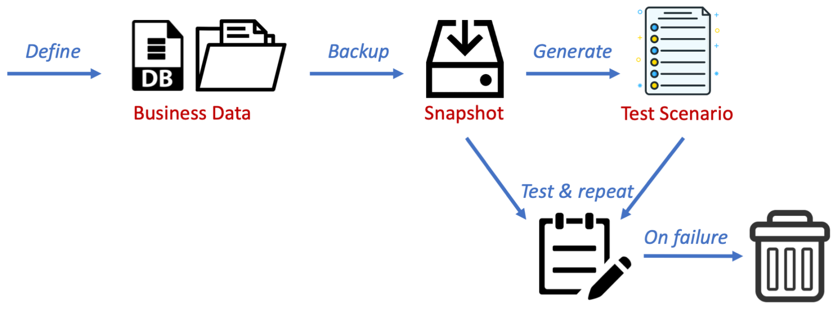

Fig. 1. Overview of Approach.

This paper proposes a framework where snapshots can be automatically tested for their usability, and for restorability with respect to target environments. The framework currently works on snapshots of Excel documents, Postgres and MySQL databases and flat files (standard directories). An overview of our approach is shown in Figure 1. The key features are the following:

- The business defines the crucial data files and databases that must survive.
- When a snapshot is created, a test scenario is generated for the backup data. An example test for an Excel file could verify the number of sheets in the document or the values of specific cells. The test for a database can verify the number of tables, column names and the values of certain database entries.
- A snapshot's usability is verified by loading the snapshot into a folder and running the tests. In the case of MySQL and Postgres, where the snapshot must be processed before running the tests, the snapshot is loaded into a Docker environment. The environment can be thrown away after running the test suite. The snapshot is considered usable only if all tests pass.
- We test the restorability of the snapshot by tweaking different properties of the Docker environment, e.g., by replacing Ubuntu 18 by Ubuntu 20 to test for restorability to a more recent Ubuntu environment.

Our aim is a simple automated framework that can be used as easily as possible. This means minimal competence to use the tool. For instance, the tests generated can mostly be generated automatically, even though the data manager can extend the suite with further tests. The approach is inspired by automated testing frameworks used in software development environments, e.g., [5, 8].

The remainder of this paper is organized as follows. We delve more deeply into the backup challenge in Section 2. We present the data testing model in Section 3 and its implementation in Section 4. Related work is described in Section 5. Conclusions and future work are presented in Section 6.

2 Backup and Restoration Challenges

We begin by clarifying the terminology used in the remainder of this paper:

- **Business data** refers to company data under normal operation. This data must survive a physical or cyber-emergency, so needs to get backed up and safely restored. Business data is only and precisely that data of value to a company in files and databases, so generally is a subset of the all data owned.
- A **snapshot** is a copy of business data for restoration. A restoration is made from a snapshot, so the snapshot must be stored separately from business data. In this way, damage to business data in an emergency does not impact snapshots. For instance, a ransomware that infects business data must not infect snapshots.
- A **backup** is a verb capturing the act of creating a snapshot from business data.
- A **restoration** is the act of replacing business data with a snapshot.
- An **archive** designates historical or non-current data. We do not consider archives here. Archives are generally read-only, stored on different media types and perhaps in different formats. The role of an archive is to index some historical event in the company's history, such as an old tax declaration, and not to provide data for use in a restoration following an incident.

Backup operations are universally considered essential to operational security in every company [13]. However, backups are only part of the story. The real challenge is to execute a successful restoration. Notably, backups are done under normal company operation. If a backup fails, it can simply be restarted. In contrast, a restoration is done in an emergency, and if it fails, the backup operation was all for nothing [17]. Companies therefore require simple frameworks to support restoration.

When a company manages its backups, restorations might fail for three reasons.

R1 The backup operation fails, so there is no usable data to restore.
R2 The snapshot becomes unusable.
R3 The environment in which the restoration is made cannot accommodate the snapshot.

Reason **R1** is the failure of the backup operation. This can be due to a configuration error (related to permissions or access paths), a storage or network error, write during copy, etc [1]. Another common reason for backup failure is human error, or where a company forgets or postpones the backup [20].

Concerning Reason **R2**, a snapshot is unusable if it cannot be read during a restoration. This happens if the snapshot data gets infected by a cyber-attack, if the storage media gets lost or damaged, or if the credentials to access a backup service where the snapshot is stored get lost or expire.

Definition Snapshot *survivability* is the property that the snapshot is usable for a subsequent restoration operation.

Some infrastructures test survivability by simply creating a message digest of a newly created snapshot; verifying that the snapshot digest remains unchanged is considered sufficient for survivability. However, this approach does not catch all errors. For instance, the message digest is not available on a cloud service that stores the backups. Also, the initial snapshot might be erroneous so the message digest is in fact a digest of invalid data. Finally, the approach requires us to be able to securely store the message digest.

Reason **R3** for restoration failure is that the environment in which the restoration is made cannot accommodate the backup. An extreme example of this is a restoration to a different environment to the one in which the backup was created, e.g., a company forced to move after an incident at its main office, and where the backup office has a different network or software infrastructure. Automating tests in these scenarios is hard. Nonetheless, we can test for restoration environments where software versions have changed since the backup environment.

Definition Snapshot *restorability* with respect to an environment E is the property that the snapshot can be restored within E.

A backup and restoration model for SMEs must be applicable to a variety of small company environments. Their infrastructures can be basic but diverse. The storage devices used include cloud boxes, hard disks, USB keys and external drives, as well as Cloud services (e.g., Dropbox, GoogleDrive, etc.). The applications that generate the data to be backed up include classical office applications (e.g., MS Excel and Word), databases (e.g., MySQL and MongoDB in cases where the database is part of a full-stack solution), flat files and folders, as well as custom business applications such as a payroll application or applications linked to company equipment (e.g., a dentist's radiography machine's application). The implication of this is that snapshots should be file based (rather than server images), that space considerations are important, and that snapshot testing must be neutral with respect to the storage media chosen for snapshots.

There are also emerging financial motivations for companies to achieve survivability and restorability. The first is the EU General Data Protection Regulation, which requires companies holding personal data belonging to EU citizens to implement secure backups. A second motivation is cyber-insurance. Following a cyber or physical incident, cyber-protection insurance contracts can pay for restorations. Therefore, it is important to make the restoration as reliable as possible to reduce costs for insurers and reduce insurance premiums for companies.

3 Model

Our goal is to automate the testing of business data. Implicit in this approach is the idea that the *data manager* – the person in the company responsible for the data – first identifies that data of high value to the company, and whose survival needs to be assured, c.f., Figure 1. After snapshot creation, a *test scenario* is generated by the data manager.

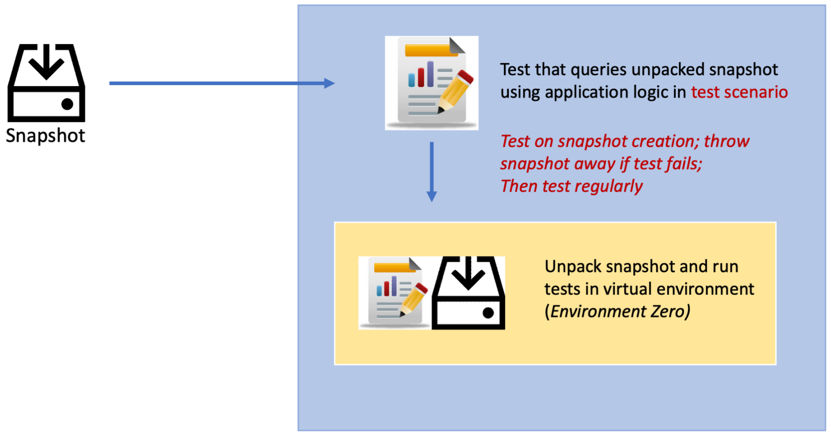
Snapshot
Test that queries unpacked snapshot using application logic in test scenario
Test on snapshot creation; throw snapshot away if test fails; Then test regularly
Unpack snapshot and run tests in virtual environment (Environment Zero)

Fig. 2. Overview of Approach.

The snapshot can be tested for survivability just after it is created, to ensure that the backup operation was successful, and at regular intervals to ensure that the snapshot is still accessible from its repository. Restoration tests, where we test that data is usable in changed environments, can be done at regular intervals.

3.1 Survivability Testing

The natural way to test snapshot survivability is simply to restore the snapshot to a temporary environment and to examine the restored contents. This approach is taken here, where automated tests check the contents of restored data, *c.f.*, Figure 2.

A shared folder is created in which the snapshot is copied and the tests are run. For MySQL and Postgres databases, an initialization phase transforms the snapshot into a queryable data structure. For this reason, the tests are run in a Docker-based virtual environment with the snapshot folder being shared with Docker.

The environment created for the survivability test is defined in a Hashicorp Language based *scenario* file, and an example is given in Figure 3. The locally_data_dir element defines where the snapshot being tested is stored.

```
module = "world"

version = "0.1.0"

local_data_dir = "backup"

environment database {
        resource mysql {
                image = "mysql:5.7.33"
                envs = {
                        MYSQL_ROOT_PASSWORD = "root"
                        MYSQL_DATABASE = "world"
                }
                ports = {
                        mysql = 3306
                }
              }
            }

```

Fig. 3. Test Scenario Description in HCL.

The image clause specifies the application image loaded into the Docker environment, which is a MySQL database in this example. The default Docker image for MySQL uses Debian Buster-Slim and PostgreSQL uses either Alpine Linux or Debian. Our default Docker container runs an Alpine Linux operating system.

The test framework is inspired by automated test frameworks used in modern software development environments like Java and Ruby/Rails, e.g., JUnit [2], RSpec [8], etc. The framework currently permits tests to be written that verify the contents of MS Excel and its OpenOffice cousin, as well as PDF documents, MySQL and Postgres dumps, as well as flat files.

The basic test operators of our framework are given in the Tables 1, 2, 3 and 4. A *test scenario* is written using these operators – and the expected contents of the snapshot. An example of tests written for an SQL snapshot is shown in Figure 4; a PDF file test is in Figure 5. These simply examine contents of the file (e.g., database entries) or examine structure (e.g., author names, names of columns or tables, etc.).

3.2 Restoration Testing

The survivability tests for databases restore data to a minimal environment, created within a Docker virtual container running with an Alpine or Buster Debian Linux. As we reject newly created snapshots that do not pass survivability tests, we know

Table 1. Operation list for MySQL

Test	Description
AssertColumnsExist(table, column_names)	Verify columns in table
AssertColumnMatch(table, column_name, regex)	Verify column values match regex
AssertRowCount(table, min, max)	Verify row count in range
Query(query, return_values)	Query table for specified values

Table 2. Operation list for Excel

Test	Description
AssertSheetsExist(sheet_names)	Check for sheet names
AssertCellsMatch(sheet, range, expr, type)	range can be A1;B5,
	type can be value, formula or link
AssertRowCount(sheet, min, max)	Verify that row count is in range
AssertColumnCount(sheet, min, max)	Verify that column count is in range

Table 3. Operation list for PDF files

Test	Description
AssertPageCount(min, max)	Number of document pages
AssertImagePageCount(page, min, max)	Number of images on page
AssertHasWatermark()	Document watermark
AssertTitleMatch(regex)	Document title
AssertAuthorMatch(regex)	Document author
AssertContentMatch(page, regex)	Content match to regex on page

Table 4. Operation list for folders

Test	Description
AssertFileCount(min, max)	Count the number of pages in the document
AssertFileNameMatch(regex)	Match file names to regex expressions
AssertFileExists(name)	File exists?
AssertFileSize(name, min, max)	File size
AssertDirExists(name)	Directory exists?

```
// Valid the structure and consistency of city table.
export function testCityTable() {
  mysql.assertColumnsExist(this, 'city', ['ID', 'Name', 'CountryCode', '
    District', 'Population'])
  mysql.assertRowsCount(this, 'city', 4000, 4500)
  mysql.assertColumnMatch(this, 'city', 'Population', '[0-9]+')
}

// Valid the structure and consistency of country table.
export function testCountryTable() {
  const columns = [
    'Code', 'Name', 'Continent', 'Region',
    'SurfaceArea', 'IndepYear', 'Population', 'LifeExpectancy',
    'GNP', 'GNPOld', 'LocalName', 'GovernmentForm',
    'HeadOfState', 'Capital', 'Code2'
  ]
  mysql.assertColumnsExist(this, 'country', columns)
  mysql.assertRowsCount(this, 'country', 239, 239)
  mysql.assertColumnMatch(this, 'country', 'Code', '^[A-Z]{3}$')
}

// Valid the structure and consistency of language table.
export function testLanguageTable() {
  mysql.assertColumnsExist(this, 'countrylanguage', ['CountryCode', 'Language
    ', 'IsOfficial', 'Percentage'])
  mysql.assertRowsCount(this, 'countrylanguage', 980, 1000)
  mysql.assertColumnMatch(this, 'countrylanguage', 'IsOfficial', '^[^TF]*(T|F
    ){1}[^TF]*$')
}
```

Fig. 4. Example Tests for a MySQL database.

```
export function setup() {
  document = new pdf.Open('paper.pdf')
}

export function testPageCount() {
  document.assertPagesCount(this, 10, 15)
}

export function testWatermark() {
  document.assertHasWatermarks(this)
}

export function testAuthorMatch() {
  document.assertAuthorMatch(this, 'Sylvain Muller and Ciaran Bryce')
}

export function testTitleMatch() {
  document.assertTitleMatch(this, 'Automated Testing of Data Survivability and Restorability')
}

export function testImageCount() {
  document.assertImagesCount(this, 2, 1, 1)
}
```

Fig. 5. Example Tests for a PDF file

that there does exist a minimum, *Environment Zero*, in which data can be restored. Further, this environment is reproducible over time since it is created from the same set of configuration parameters.

Restoration testing is really just a special case of survivability testing. The key feature with restoration testing is to test with different software environments and versions, e.g., testing restoration to an older or future version of the database, or to a different operating system. This is done by defining a different HCL scenario file with a different Docker image.

Concretely, a Docker image file is downloaded from hub.docker.com using the Dockerfile that specifies the environment. For instance, the Dockerfile could specify "FROM mysql:8.0.18", as in the Dockerfile of Figure 6, meaning that the restoration test is made with version 8.0.18 of MySQL. The test can be replicated with different scenarios for different software environments. For instance, if MySQL 10.0 is made available, a new agent can be created with a Dockerfile with "FROM mysql:10.0".

The Docker framework can test restoration with different environmental settings. This is done by creating a shared folder between the calling environment and Docker in which system files can be transferred. This allows us to test the restoration against these new system files. For instance, consider the command: *docker run –name=psql -U softpeel -d -v /pgdata:/var/lib/pgsql/data -p 5000:3306 psql*. This runs the Postgres server in a Docker environment with the particularity that the

```
FROM golang:alpine as builder

WORKDIR /app

RUN mkdir -p /data
RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build github.com/tigerwill90/
    replicas/cmd/mysql

FROM mysql:8.0.18

ENTRYPOINT ["/mysql"]
EXPOSE 5000
```

Fig. 6. An edited Dockerfile importing an Ubuntu OS image.

configuration folder in the Docker environment is replaced by the psgdata folder in the host environment.

3.3 Discussion

We believe that the framework presented here is quite flexible with respect to how it integrates into a company's data backup process.

When should snapshots be made? We do not specifically address this question in our framework. Our aim is simply to make it possible to test the snapshots. Issues such as the frequency of snapshot creation, conservation duration, number of snapshots, etc., is a business decision that company owners must make for themselves based on the value of their data and calculated risks to data. On the other hand, the framework does not impose any particular rhythm on the backup operations.

Who writes tests, and when? A test suite for a snapshot should be generated any time after the snapshot is created. A good rule of thumb is to write and run the test suite just after snapshot creation since this permits detection of snapshots corrupted during creation.

As suggested in the examples in Figures 4 and 5, our aim is to support the automatic generation of tests from business data. This is because most tests simply examine random content from the business data, or verify structure like the number of sheets in an Excel file, the names of tables and columns in a Postgres database, etc. On the other hand, a data manager is able to extend the test suite with his own tests. This approach is now common practice in automated software testing [8].

What should be done when a test suite fails? A test suite failure means that the snapshot cannot be restored in the environment described in the Docker-based environment. If this is the current working environment of the organization, then this means that the snapshot is no longer usable – unless some environment can be found and tested that permits restoration.

In reality, a test failure underlies the advantage of our model, since it pinpoints snapshots that cannot be restored before an emergency where a real restoration from the snapshot is needed.

How does the testing framework integrate in the backup process? Snapshot testing is an independent brick in the process of saving and restoring business data. Tests can be launched manually or automatically via a cron job. Tests can also be triggered via REST calls to our test motor's HTTP server. The latter allows us to integrate testing into an automated backup process based on the open-source Restic framework (see *Restic.net*). Restic automates the copy of file repositories that include both local folders as well as Cloud providers (e.g., Amazon, Google Cloud, MicroSoft Azure), FTP servers and REST-enabled Web servers.

What is particular about restoration tests? Fundamentally, there is no real difference between a survivability and restoration test from the perspective of using the framework. Only the restoration environment described in the scenario file changes.

An ideal situation is where we can test that snapshots can be restored in a wholly different environment, as can happen when a company foresees the need to change location after a major incident or when a software migration is being prepared for at the local site. Testing for major environmental changes is a challenge, since there are so many parameters to consider: machine hardware and network configurations, software licenses, user permissions, etc. This is beyond the scope of our current framework, but does merit further research.

4 Implementation

The framework is a *motor* written in Golang that runs on Linux, Mac OS X and Windows. The motor is accessed via a REST API that also includes encapsulates a command line interface. The only external dependency of the tool is with Docker[1] which provides the virtual containers in which survivability and restorability tests are run on database snapshots. This has the advantage that the tests do not have a side effect on the rest of the system. Further, the snapshot itself does not get modified in the real environment; only in the virtual environment. In this way, the snapshot is immutable for its whole lifetime.

The tests are written in JavaScript; the template is shown in Figure 7. Test programmers simply add the tests they require to the test scenario. Added test functions simply need to begin with "test" and respect Camel casing.

A test can be run from the command line and specific tests can also be executed:

[1] www.docker.com

```
import { sleep } from "scenario";

// Setup allow to do extra setup before testing like recover a database
export function setup() {
        console.log('recovering date')
        sleep(30)
}

// Each test name must begin by the suffix 'test'.
export function testShouldFail() {
        this.fail('failed')
}

// Teardown allow you to perform clean up operation
export function teardown() {
        // some clean up
}
```

Fig. 7. JavaScript test template.

– scenario run – run all test methods in the suite
– scenario testShould[A-z]+$ – run all test methods that match the argument.

The setup method reads the data from the folder where the snapshot is stored. The example of Figure 8 comes from the test of a MySQL database.

The framework is written in just over 10 thousand lines of Golang code. The kernel of the framework implements assertions for database and other content types, as well as the link to Docker for starting the test environment.

From a performance perspective, the majority of the execution time is on loading the data into the Docker environment. For instance, the MySQL example of Figure 4 takes 30 seconds on a 400 kB database. This includes the time to create the Docker container with the required image, unzip the database, load it into the container and then run the tests. The time is comparable to that needed to start a database server in a production environment.

5 Related Work

All standard operating systems come with tools for backup and restoration [14]. Windows for instance allows for incremental and differential backups of the file system. Files and folders possess an archive flag that is set whenever the file is modified, and in need of backup. This bit is reset when an incremental or full backup is made. The idea behind this approach is to facilitate a full system restoration. In contrast, the goal of our approach is to permit an SME to identify a critical set of business data that needs to be restorable, and then to handle the backup and restoration of just that data. This allows the SME to optimize backup storage and to focus on the company-critical data. Further, our method supports restoration in environments different to those in which backups are done.

```
import { sleep, retry } from "scenario"
import archive from "scenario/archive"
//import { unzip } from "scenario/archive"
import docker from "scenario/docker"
import sql from "scenario/assert/sql"

const config = {
  user: 'softpeel',
  password: db.postgres.envs.POSTGRES_PASSWORD,
  db_name: db.postgres.envs.POSTGRES_DB,
  port: db.postgres.getPort('postgres'),
  ip: db.postgres.getIp()
}

const postgres = new sql.Client('postgres', config)

export function setup() {
  // Exponential backoff retry that ping world database it is ready
  retry(function () {
    return postgres.ping()
  }, 30)

  const zip = archive.Zip('softpeel.sql.zip', {overwrite: true})
  zip.extract()
  const f = open('softpeel.sql')

  // MySQL cli config to recover sql dump.
  const config = {
    cmd: ['psql', '-U', 'softpeel', '-d', db.postgres.envs.POSTGRES_DB],
    envs: [`POSTGRES_PASSWORD=${db.postgres.envs.POSTGRES_PASSWORD}`],
    stdin: f,
  }

  // Execute command on mysql resource
  const code = docker.exec(db.postgres, config)
  if (code != 0) {
    throw ('recovery fail')
  }
}
```

Fig. 8. The setup method creates reads the data for the enclosing Docker environment

With the advent of the cloud, an emerging approach to handle backups is to save a snapshot of each virtual machine's current image, e.g., [19]. This contrasts with the file-based backup because all application code and data is backed up. Most cloud providers implement this technique. There are several challenges nonetheless. First, the image is not certain to have been taken at a moment when the data is consistent; for instance, a transaction may have been running on the database. Second, the solution requires a lot of storage space, which makes indexing and deduplication harder, e.g., [7]. In contrast, the file-based approach such as the one taken in this paper encourages users to identify business critical data, is more economical in storage space, and snapshots can be restored to a wider range of environments.

Recent research in data storage generally looks at issues like storage area networks design, performance and security, e.g., [6, 1], backup techniques, e.g., [9], or specific angles of backups like minimizing service outage during backups [18, 11] or deduplication [12]. However, the issues of survivability and restorability are less frequently treated. This needs to change as regulations increasingly require companies to conserve data and comply to external requests for data conservation.

Frameworks with scripts that automate backups have appeared, e.g., *restic.net*, and are accessible to SMEs. This is a positive development since automating the backup and restoration procedure reduces the margin of error in the operations. We have used our test framework in Restic scripts where the tests are invoked in the script after the snapshot is created., but it can be added to any automated backup system.

A related field is *infrastructure-as-code* (IaC), e.g., [3, 15, 10]. IaC is the DevOps [16] idea where developers and operators use automated scripts for the provision of software systems in their company. IaC greatly facilitates software system updates since the process is made as efficient as possible. Importantly, it is easier to do a *rollback* if the system is discovered to have post-deployment bugs since the steps taken by human operators are removed or reduced. IaC was invented for code deployment but the principles are just as applicable to data backups and restorations. Further, it is natural to be able to add a test framework such as the one presented in this paper to these IaC scripts since data, like code and services, are fast-class entities in company infrastructures.

6 Conclusions

This paper has presented a framework for the methodical testing of file-based data backups. The file-based approach optimizes backup storage space and allows companies to focus on their critical business data. The approach can also be used by companies that run virtual servers on the cloud, even though the cloud provider creates backup snapshots of the virtual machines. Storing data in the cloud does not remove responsibility from companies to control and test their backed up data.

Automated testing works for MicroSoft and Open Office (spread-sheet and document) files, MySQL exports, flat files and PDF documents. Future work will extend test coverage to image files and encrypted files. To test encrypted snapshots, we need to extend the framework to run the tests in a sandboxed environment, such as that supported by Linux AppArmor, and for which Docker can be run [21].

References

1. George Amvrosiadis and Medha Bhadkamkar. Getting back up: Understanding how enterprise data backups fail. In *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016.*, pages 479–492, 2016.
2. Yoonsik Cheon and Gary T. Leavens. A simple and practical approach to unit testing: The JML and junit way. In *ECOOP 2002 - Object-Oriented Programming, 16th European Conference, Malaga, Spain, June 10-14, 2002, Proceedings*, pages 231–255, 2002.
3. Clauirton de Siebra, Rosberg Lacerda, Italo Cerqueira, Jonysberg P. Quintino, Fabiana Florentin, Fabio Q. B. da Silva, and André L. M. Santos. From theory to practice: The challenges of a devops infrastructure as code implementation. In *Proceedings of the 13th International Conference on Software Technologies, ICSOFT 2018, Porto, Portugal, July 26-28, 2018.*, pages 461–470, 2018.
4. Vasiliki Diamantopoulou, Aggeliki Tsohou, and Maria Karyda. From ISO/IEC 27002: 2013 information security controls to personal data protection controls: Guidelines for GDPR compliance. In Sokratis K. Katsikas, Frédéric Cuppens, Nora Cuppens, Costas Lambrinoudakis, Christos Kalloniatis, John Mylopoulos, Annie I. Antón, Stefanos Gritzalis, Frank Pallas, Jörg Pohle, M. Angela Sasse, Weizhi Meng, Steven Furnell, and Joaquín García-Alfaro, editors, *Computer Security - ESORICS 2019 International Workshops, CyberICPS, SECPRE, SPOSE, and ADIoT, Luxembourg City, Luxembourg, September 26-27, 2019 Revised Selected Papers*, volume 11980 of *Lecture Notes in Computer Science*, pages 238–257. Springer, 2019.
5. Thomas Fehlmann and Eberhard Kranich. A framework for automated testing. In Murat Yilmaz, Jörg Niemann, Paul M. Clarke, and Richard Messnarz, editors, *Systems, Software and Services Process Improvement - 27th European Conference, EuroSPI 2020, Düsseldorf, Germany, September 9-11, 2020, Proceedings*, volume 1251 of *Communications in Computer and Information Science*, pages 275–288. Springer, 2020.
6. Kazuo Goda. Storage area network. In *Encyclopedia of Database Systems, Second Edition.* 2018.
7. Jianxin Li, Yangyang Zhang, Jingsheng Zheng, Hanqing Liu, Bo Li, and Jinpeng Huai. Towards an efficient snapshot approach for virtual machines in clouds. *Inf. Sci.*, 379:3–22, 2017.
8. Myron Marston and Ian Dees. *Effective Testing with RSpec 3: Building Ruby Apps with Confidence.* The Pragmatic Bookshelf, 2017.
9. Donghee Min, Taegye Hwang, Joonhyouk Jang, Yookun Cho, and Jiman Hong. An efficient backup-recovery technique to process large data in distributed key-value store. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 2072–2074, 2015.
10. Kief Morris. *Infrastructure as Code.* O'Reilly, 2016.
11. Pratik Mukherjee and Valentina Salapura. Challenges of DB2 restore in a distributed systems environment and engineered solutions. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2018, Luxembourg, June 25-28, 2018*, pages 38–42, 2018.
12. J. K. Periasamy and B. Latha. Efficient hash function-based duplication detection algorithm for data deduplication deduction and reduction. *Concurr. Comput. Pract. Exp.*, 33(3), 2021.

13. Antònia Mas Picahaco, Antoni Lluís Mesquida, Esperança Amengual Alcover, and Bartomeu Fluxà. ISO/IEC 15504 best practices to facilitate ISO/IEC 27000 implementation. In *ENASE 2010 - Proceedings of the Fifth International Conference on Evaluation of Novel Approaches to Software Engineering, Athens, Greece, July 22-24, 2010*, pages 192–198, 2010.
14. W. Curtis Preston. *Backup and recovery - inexpensive backup solutions for open systems: covers Windows, Linux, Unix, and OS X*. O'Reilly, 2007.
15. Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. A systematic mapping study of infrastructure as code research. *Information & Software Technology*, 108:65–77, 2019.
16. James Roche. Adopting devops practices in quality assurance. *Commun. ACM*, 56(11):38–43, 2013.
17. Jibran Saleem, Bamidele Adebisi, Ruth Ande, and Mohammad Hammoudeh. A state of the art survey - impact of cyber attacks on sme's. In *Proceedings of the International Conference on Future Networks and Distributed Systems, ICFNDS 2017, Cambridge, United Kingdom, July 19-20, 2017*, page 52, 2017.
18. Xiaoyan Yin, Javier Alonso, Fumio Machida, Ermeson C. Andrade, and Kishor S. Trivedi. Availability modeling and analysis for data backup and restore operations. In *IEEE 31st Symposium on Reliable Distributed Systems, SRDS 2012, Irvine, CA, USA, October 8-11, 2012*, pages 141–150, 2012.
19. Lingfang Zeng, Shijie Xu, and Yang Wang. Vmbackup: an efficient framework for online virtual machine image backup and recovery. *Concurr. Comput. Pract. Exp.*, 28(9):2630–2643, 2016.
20. Lingfang Zeng, Shijie Xu, and Yang Wang. Data backup: Do business want to measure recovery potential? *Issues in Information Systems*, 19(1):20–28, 2018.
21. Hui Zhu and Christian Gehrmann. Lic-sec: an enhanced apparmor docker security profile generator. *IACR Cryptol. ePrint Arch.*, 2020:1147, 2020.