A Pilot Study of Testing Infrastructure as Code for Cloud Systems

Nabhan Suwanachote*, Soratouch Pornmaneerattanatri†, Yutaro Kashiwa†, Kohei Ichikawa†, Pattara Leelaprute*, Arnon Rungsawang*, Bundit Manaskasemsak*, Hajimu Iida†,

*Kasetsart University, Thailand — †Nara Institute of Science and Technology (NAIST), Japan

***Abstract*—Infrastructure as Code (IaC) has become the de-facto standard method for managing cloud resources. Just like general source code (*e.g.,* Java, etc.), infrastructure code also has numerous bugs so it needs to be tested. While several testing frameworks for IaC for cloud systems have been developed in practice, researchers have paid little attention to their testing. This study presents an empirical investigation of the use of tests for IaC for cloud systems.**

Our empirical results show that (i) 55.2% of the repositories using Terratest have at least one server infrastructure test; (ii) developers often maintain server infrastructure tests (1.7%-11.3% commits out of all the commits); (iii) many repositories have tests for system functionality (28%), deployment (20%), and configuration (17%).

***Index Terms*—Infrastructure as Code, Software Testing, Cloud System**

I. Introduction

Infrastructure as Code (IaC) has become the de-facto standard method for managing cloud resources. For example, Kubernetes[1] can manage containers on cloud servers using YAML setting files (*i.e.,* IaC). The CNCF's annual report in 2022 says that 96% of organizations are either using or evaluating Kubernetes, and over 5.6 million developers are estimated to be the users [1]. Also, it is reported that the global market size of Infrastructure as Code (IaC) is expected to reach 2.8 billion dollars by 2028 [2].

IaC for cloud systems helps developers automate most of the deployment process and configuration changes. For example, cloud systems can automatically modify the number of servers according to the amount of server accesses. IaC mitigates efforts to manage cloud systems and reduce the chances of human errors.

However, infrastructure code may have bugs, just as general source code does (*e.g.,* Java, etc). Rahman *et al.* [3] examined defect-related commits modifying Puppet[2] scripts which aims to set up local servers. They reported that there are numerous bugs in such scripts. Puppet is not a cloud management tool but is still a type of IaC tool. Therefore, IaC for cloud systems would also contain bugs and need to be tested. In fact, several testing frameworks for infrastructure code for cloud systems have been developed in practice.

In spite of the growth in the population of IaC for cloud systems and the demands of testing them, researchers have paid little attention to their testing. It is thus still unclear how software developers practice testing infrastructure code for cloud systems and evolve their test scripts. In this study, our goal is to understand:

"How do developers test infrastructure code for cloud systems and evolve the test scripts?"

To answer this central question, this study presents an empirical investigation on the use of tests for infrastructure code for cloud systems. Specifically, we investigate (i) the number of infrastructure tests, (ii) the amount of effort to maintain infrastructure tests (*i.e.,* modifications), and (iii) the contents of tests. This study examines repositories using one of the popular testing frameworks for cloud systems, Terratest.[3] While Terratest was originally developed for Terraform, it now supports testing functions for Kubernetes, AWS, etc. The repository of Terratest has already received more than 7,000 stars as of August 2023. Studying tests with the popular testing framework would help understand how developers test infrastructure code for cloud systems.

II. Study Design

This section describes the overview of data collection and data analysis processes to answer our research questions.

A. Data Collection

To study tests for cloud systems, we need to find the test code contained in repositories. However, it is difficult to identify the test code because there are no de facto standard testing tools for cloud systems. While there are several tools, we shed light on Terratests because it is a popular tool receiving more than 7,000 stars.

Terratest is one of the popular infrastructure testing libraries written in Go language. It provides testing functions for authorization, deployment, clean-up, and any configuration changes in real cloud environments. Terratest offers multiple modules that can work with tools and platforms: Terraform, Kubernetes, AWS, GCP, Azure, etc. Terratest can be used with Go's built-in testing package and other Go testing frameworks.

To identify the repositories using Terratest, we searched GitHub repositories for Go module files that declare Terratest modules. Specifically, to find the repositories written in Go, we employed GitHubSearch [4]. Using this tool, we searched

[1]https://kubernetes.io/

[2]https://www.puppet.com/blog/what-is-infrastructure-as-code

[3]https://terratest.gruntwork.io/docs/

TABLE I
DATASET DESCRIPTION

	Min	Mean	Median	Max
Number of Commits	49	1,217	591	21,833
Number of Test cases	1	144	58	864
Development duration (month)	3	42	41	105
Line of Code	845	146,256	19,719	2,030,197

for non-forked Go projects. In addition, to assure the quality of repositories, we excluded repositories that do not meet the following conditions:

Criterion 1: Must have at least five developers (*i.e.,* team development [5])
Criterion 2: Must have been developed for over one month (*i.e.,* enough development time [5])
Criterion 3: Must have at least two commits per month (*i.e.,* must be active [6])

With these conditions, GitHubSearch returned 14,192 repositories. After that, we cloned all the repositories and parsed their go module files (*i.e.,* go.mod) on the main/master branch to find the repository using Terratest. As a result, we found 58 public repositories using Terratest modules (and actually using Terratest methods in their test code). While the number of the studied repositories is not large, we believe that this pilot study helps us understand the practice of testing IaC for cloud systems and our future work will investigate repositories using other testing libraries/frameworks.

Table I provides an overview of our dataset, and Figure 1 shows the number of repositories using each module. The largest number of repositories in the dataset is those using Terraform, followed by those using Kubernetes and Helm.

B. Test case identification

After identifying the repositories, we first need to find test cases in their repositories. To do so, we employ an abstract syntax tree (AST) from Go's AST package.[4] After that, we detect test cases using two criteria: 1) the file containing test cases must end with "_test.go"; 2) the function name must start with "Test". These two criteria allow us to detect test cases written in Go's standard testing package style.

Then, we classify the test cases into server infrastructure tests and local infrastructure tests. Server infrastructure tests are test methods that provision cloud resources in a real environment. Whereas, local infrastructure tests do not provision any cloud resources in a real environment (*e.g.,* check if YAML files are correctly parsed). The reason why we distinguish them is that we are more interested in test cases verifying the manipulation of cloud resources. Note that local and server infrastructure tests are often considered integration and unit tests, respectively [7]. However, we decided to use different names so that server and local infrastructure tests can be distinguished from other tests (*i.e.,* for applications).

[4]https://pkg.go.dev/go/ast

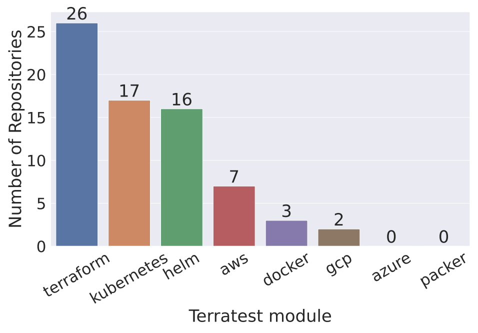

Fig. 1. Number of repositories testing each IaC system

To find server infrastructure tests, we consider the test cases using the following methods in Terattest's library as server infrastructure tests:

Terraform[5]**:** *Apply, InitAndApply, Destroy, etc.*
Kubernetes[6]**:** *KubectlApply, KubectlDelete, etc.*
Helm[7]**:** Install, InstallE, Delete, etc.

These methods access cloud resources, so we decided to classify them in server infrastructure tests. The complete list is shown in our replication package.[8] In addition, we classify test methods that use the other Terratest's methods as local infrastructure tests, and test methods that use no Terratest's methods as other tests.

C. Data Analysis

This study answers three research questions based on the number of infrastructure tests (RQ1), the effort to maintain tests (RQ2), and the contents of tests (RQ3). In the following, we describe the methodologies that are used to answer each research question.

RQ_1: What percentage of tests are for testing infrastructure?

RQ1 first examines the number of server, local infrastructure tests, and other tests for each project. We then measure the median percentage of server and local infrastructure tests out of all test cases across projects.

RQ_2: How often are Infrastructure test cases updated?

To study the frequency of updates to each type of test, we first clone repositories and check out each revision. We then identify if each test case is server, local infrastructure tests, or other tests, as described in Section II-B. We obtain git diff for each revision and find if each test method is modified in this commit. We applied this process to all revisions.

However, this process requires high computation resources and time. We thus decided to study the commits that were made during two years (between Aug 1, 2021 and July

[5]https://pkg.go.dev/github.com/gruntwork-io/terratest@v0.43.12/modules/terraform
[6]https://pkg.go.dev/github.com/gruntwork-io/terratest@v0.43.12/modules/k8s
[7]https://pkg.go.dev/github.com/gruntwork-io/terratest@v0.43.12/modules/helm
[8]https://github.com/nabhan-au/Testing-IaC-Research/blob/main/terratest_integration_func.csv

TABLE II
NUMBER OF COMMITS MODIFYING EACH TYPE OF TEST

	Number of commits modifying			
	All (including source)	Local infrastructure tests	Server infrastructure tests	Other tests
sumologic	1,161	28 (2.4%)	20 (**1.7**%)	6 (0.5%)
k8gb	465	15 (3.2%)	13 (**2.8**%)	9 (1.9%)
terragrunt	225	2 (0.9%)	4 (**1.8**%)	98 (43.6%)
helm-charts	188	6 (3.2%)	8 (**4.3**%)	0 (0.0%)
cloud-nuke	115	8 (7.0%)	13 (**11.3**%)	85 (73.9%)

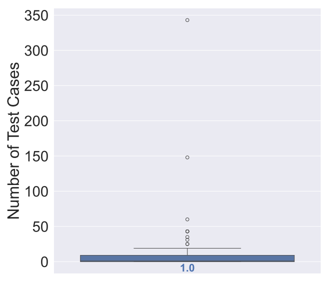

Fig. 2. Number of local infrastructure tests

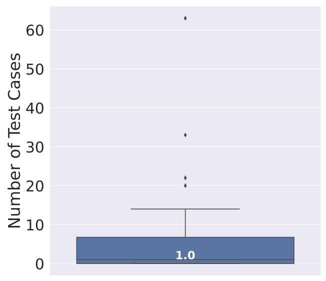

Fig. 3. Number of server infrastructure tests

31, 2023) from five projects that were randomly selected (cloud-nuke[9], terragrunt[10], k8gb[11], helm-charts[12], sumologic-kubernetes-collection[13]). The process required a total of more than 1,000 computation hours in servers with 20 CPU cores.

RQ_3**: What test cases are made for infrastructure tests?** We randomly chose 100 server infrastructure test methods used in RQ1 and performed open coding [8] to classify what is tested. Specifically, two of the authors manually and independently inspect the test cases. If there are conflicts between the two inspectors, another author joins the manual inspection and discusses among three inspectors to resolve the conflicts. Note that two of the inspectors have more than 10 years of software development experience, and one has 3 years of software development experience but has experience developing a commercial system. After our manual inspection, there were 26 conflicts between two inspectors, which demonstrates a substantial agreement (Cohen's kappa coefficient = 0.71). These conflicts were resolved by the third inspector.

III. RESULTS

RQ_1: What percentage of tests are for testing infrastructure?

Figure 2 and 3 show the number of local infrastructure tests and server infrastructure tests, respectively. We found that repositories have a median of 1.0 for both local infrastructure tests and server infrastructure tests but average at 15.8 and 5.1, respectively.

Also, we observed that 55.2% have at least one server infrastructure test, while most repositories have only a few server infrastructure tests.

In addition, we investigated the percentages of server and local infrastructure tests out of all test cases. We found that repositories have a median of 0.5% local infrastructure tests and 0.6% of server infrastructure tests. While the average of local infrastructure tests (15.8 test cases) is greater than that of server infrastructure tests (5.1 test cases), their median values are the same (1.0 test case).

Answer to RQ1: *A median of 0.6% of tests are for testing server infrastructure, and 55.2% of the repositories using Terratest have at least one server infrastructure test.*

A. RQ2: How often are infrastructure test cases updated?

Table II shows the number of commits modifying each type of test. From the five repositories, we collected 1,139 commits in total. Out of all the commits, including source code changes, 1.7% of commits modify server tests in sumologic (*i.e.,* 20 commits), 2.8% in k8gb (*i.e.,* 13 commits), 1.8% in terragrunt (*i.e.,* 4 commits), 4.3% in helm-charts (*i.e.,* 8 commits), and 11.3% in cloud-nuke (*i.e.,* 13 commits). Interestingly, the results suggest that server infrastructure tests are often maintained while the repositories do not have many server infrastructure tests.

Answer to RQ2: *Developers often maintain server infrastructure tests (1.7%-11.3% out of all the commits).*

B. RQ3: What test cases are made for infrastructure tests?

We manually and independently categorized 100 server infrastructure tests. Throughout our manual inspection, we observed seven categories in total. In the following, we introduce each type of server infrastructure test.

System Functionality: This kind of test verifies software functions in the real environment. We found 28 test cases (28%) classified into this category. In Snippet 1, the test checks the output of their function "GetSecretManagerSecretString".

```
func TestGetSecretString(t *testing.T) {
 ...
 actualSecret, err := GetSecretsManagerSecretString(opts
   , secretARN)
 require.NoError(t, err)
 assert.Equal(t, secretVal, actualSecret)
}
```

Snippet 1. An example of system functionality/deployment test

[9]https://github.com/gruntwork-io/cloud-nuke
[10]https://github.com/gruntwork-io/terragrunt
[11]https://github.com/k8gb-io/k8gb
[12]https://github.com/cockroachdb/cockroach
[13]https://github.com/SumoLogic/sumologic-kubernetes-collection

System Deployment: This kind of test verifies whether IaC can correctly deploy applications to a real environment. We observed 20 test cases (20%) classified in this category. In Snippet 2, a test case checks if their applications with a specific option (Line 2) work on the Terraform cluster.

```
1  func TestDeploymentTriggerCreation(test *testing.T) {
2    terraformTest := &terraform.Options{
3       TerraformDir: "../examples/Deployment-Trigger-
          Creation",...}
4    ...
5    if _, err := terraform.ApplyE(test, terraformTest);
6      err != nil {fmt.Println(err)}}
```

Snippet 2. An example of a system deployment test

System Configuration: This kind of test validates, manages, or updates system/infrastructure configurations in a real environment. For example, the test in Snippet 3 verifies if a wrong configuration (Line 7) is correctly identified as an error (from Line 10 to Line 13). We found 17 test cases (17%) during our manual inspection.

```
1  func TestAlertRuleCategories(t *testing.T) {
2    ...
3    invalidOptions := terraform.
      WithDefaultRetryableErrors(t, &terraform.Options{
4       ...
5       Vars: map[string]interface{}{
6          "name":           name,
7          "event_categories": []string{"INVALID"},
8          ...},})
9    _, err := terraform.ApplyE(t, invalidOptions)
10   if assert.Error(t, err) {
11      assert.Contains(t,
12         err.Error(),
13         "event_categories.0: can only be 'Compliance',
            'App', 'Cloud', ...)
14   }}
```

Snippet 3. Test system configuration

System Integration: This kind of test ensures that the entire system or modules work correctly with other services such as third-party services, cloud services, internal services, cache servers, and databases, so-called service-to-service. We discovered 16 test cases (16%) classified into this category. For example, in Snippet 4, the test initializes the service configuration (from Line 3 to Line 7) and deploys their cloud service and application. After that, the test checks if the outputs in JSON can be obtained by sending an API request (from Line 12 to Line 16).

```
1  func TestApi(t *testing.T) {
2   ...
3   dbSvc = db.New(dynamodb.New(...))
4   ...
5   usageSvc = usage.New(dynamodb.New(...))
6   sqsSvc = sqs.New(awsSession, ...)
7   codeBuildSvc = codebuild.New(awsSession, ...)
8   ...
9   t.Run("Get Accounts", func(t *testing.T) {
10   ...
11   t.Run(..., func(t *testing.T) {
12   resp := apiRequest(t, &apiRequestInput{
13     method:"GET", url:apiURL + "/accounts", json:nil,
14   })
15   results := parseResponseArrayJSON(t, resp)
16   assert.Equal(t, results, []map[string]interface{}{},
      "API should return []")})
17   ...
```

Snippet 4. An example of system integration test

System Security: This kind of test checks if the system is fully secured. We found 8 test cases (8%) classified into this category. For example, in Snippet 5, the test case checks if a secret string is appropriately deleted in the system (Line 4) and a new session does not have it (Line 9 and Line 10).

```
1  func TestNukeSecretReplica(t *testing.T) {
2    ...
3    secretName := fmt.Sprintf("test-cloud-nuke-
      secretsmanager-one-%s", random.UniqueId())
4    defer terraws.DeleteSecretE(t,region,secretName,true)
5    arn := createSecretStringReplicaWithDefaultKey(t,
      region, secretName)
6    session, err := session.NewSession(...)
7    ...
8    // Make sure the secret is deleted.
9    _, err = terraws.GetSecretValueE(t, region, arn)
10   assert.Error(t, err)
11 }
```

Snippet 5. An example of system security test

System Reliability: This kind of test ensures the system keeps working even if unexpected incidents happen. We discovered 6 cases (6%) during the manual inspection. Snippet 6 shows an example of this case. In the Snippet, the test intentionally destroys their network (Line 9) and checks if their networks failover (Line 10 and Line 11).

```
1  func TestFullFailover(t *testing.T) {
2         ...
3     instanceEU, err := utils.NewWorkflow(...).
4         WithGslb(gslbPath, host).
5         WithTestApp("eu").
6         Start()
7         ...
8     t.Run("kill podinfo on the second cluster", func(t *
       testing.T) {
9         instanceUS.StopTestApp()
10        err = instanceUS.WaitForExpected(euLocalTargets)
11        require.NoError(t, err)
12        ...
```

Snippet 6. An example of a system reliability test

System Network: This kind of test checks if networking on the system is available or works correctly. We found 5 test cases (5%) classified into this category. Snippet 7 shows an example of this case. In the Snippet, the test case checks if a newly introduced DNS setting is working correctly.

```
1  func TestFakeDNSBasic(t *testing.T) {
2   NewFakeDNS(testSettings).
3   AddNSRecord("blah.cloud.example.com.", "gslb-ns-us-
      cloud.example.com.").
4   ...
5   AddARecord("ip.blah.cloud.example.com.", net.IPv4(10,
      0, 1, 5)).
6   Start().RunTestFunc(func() {
7     g := new(dns.Msg)
8     g.SetQuestion("ip.blah.cloud.example.com.", dns.
       TypeA)
9     a, err := dns.Exchange(g, fmt.Sprintf("%s:%v",
       server, port))
10    require.NoError(t, err)
11    require.NotEmpty(t, a.Answer)
12   }).RequireNoError(t)
13 }
```

Snippet 7. An example of system network test

Answer to RQ3: *The most common test cases are about "System Functionality" (28%), followed by "System Deployment" (20%) and "System Configuration" (17%).*

IV. Discussions

A. Future Direction

Our study revealed that a non-negligible number of repositories contain server infrastructure tests but developers often maintain these tests. Still, it is not certain how and to what extent server infrastructure tests can benefit developers. In the future, in addition to expanding datasets, we plan to look into the following directions.

Effectiveness of Server Infrastructure Tests. Generally speaking, depending on the type of test such as combination tests and unit tests, the variety of detected bugs is different. However, it is not certain what types of bugs can be found by server infrastructure tests. Thus, we plan to perform manual inspections of bug reports and build failures associated with server infrastructure tests.

Optimal practice of snapshot testing. We would like to survey developers to understand best practices and anti-patterns by asking how they design server infrastructure tests and what issues they encounter.

B. Threats to Validity

Internal validity: Our manual inspection are performed by two authors in order to avoid subjectiveness bias, and a third author was involved in case of conflicts. However, we cannot exclude imprecision in the process.

External validity: Our study mainly focuses on Terratest to identify IaC tests for cloud systems. We believe that IaC tests without using Terratest are not different from those using Terratest, but still, we cannot generalize our findings.

V. Related Work

A. Finding Issues in Infrastructure as Code

There are several papers studying Issues in IaC. While these papers are similar to our study in terms of IaC, they are not about testing.

Rahman *et al.* [9] analyzed IaC scripts to identify code security-related code smells. They performed a manual inspection and found 67,801 occurrences of security smells, including 9,175 hard-coded passwords.

Chen *et al.* [10] has proposed an approach to find IaC code errors in code changes. Their results show that the approach detected 41 error patterns from 14 popular Puppet projects.

Rahman *et al.* [3] examined defects in the Puppet scripts. They categorized 1,448 defect-related commits from 291 repositories and conducted surveys with 66 practitioners to ask if they agreed with the created categories. Throughout their empirical study, they revealed that there are eight categories of defects, such as Syntax and Configuration data.

B. Testing Infrastructure as Code

Several papers are similar to our paper on studying the IaC testing. While these papers study actual bugs or testing practices summarized in gray literature, our paper is different from them because we focus on test case characteristics based on what is tested in the cases.

Hassan *et al.* [5] analyzed bugs in Ansible scripts. They conducted a manual inspection to classify bugs into eight categories. Also, they examine testing patterns and classify them into three patterns.

In their later work, Hassan *et al.* [11] perform a glay literature review to find testing practices for IaC scripts. Throughout their open coding, they identified six types of testing practice articles including "Avoiding Coding Anti-patterns", "Use of Automation", etc.

VI. Conclusion

This study examined the repositories using Terratest in order to reveal how developers test infrastructure on cloud systems. We collected 59 repositories that use Terratest modules and conducted quantitative and qualitative analyses.

Throughout our empirical study, we found that (i) 55.2% of the repositories using Terratest have at least one server infrastructure test; (ii) developers often maintain server infrastructure tests (1.7%-11.3% commits out of all the commits); (iii) many repositories have tests for system functionality (28%), deployment (20%), and configuration (17%).

Acknowledgment

We gratefully acknowledge the financial support of JSPS for the KAKENHI grants (JP21H03416, JP21K17725) and JST for the PRESTO grant (JPMJPR22P3).

References

[1] C. N. C. Foundation, "Cncf-annual-survey-2021," accessed: 2023-08-13. [Online]. Available: https://www.cncf.io/wp-content/uploads/2022/02/CNCF-Annual-Survey-2021.pdf

[2] M. I. S. P. Ltd, "Global infrastructure as code (iac) market size, share & industry trends analysis report by component, by type, by infrastructure type (mutable and immutable), by deployment mode, by organization size, by vertical, by regional outlook and forecast, 2022 - 2028."

[3] A. Rahman, E. Farhana, C. Parnin, and L. Williams, "Gang of eight: A defect taxonomy for infrastructure as code scripts," in *Proc. of the 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE'20)*, 2020, pp. 752–764.

[4] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *Proc. of the 18th IEEE/ACM International Conference on Mining Software Repositories (MSR'21)*, 2021, pp. 560–564.

[5] M. M. Hassan and A. Rahman, "As code testing: Characterizing test quality in open source ansible development," in *Proc. of the 15th IEEE Conference on Software Testing, Verification and Validation (ICST'22)*, 2022, pp. 208–219.

[6] P. Pickerill, H. J. Jungen, M. Ochodek, M. Maćkowiak, and M. Staron, "PHANTOM: Curating GitHub for engineered software projects using time-series clustering," *Empirical Software Engineering*, vol. 25, no. 4, pp. 2897–2929, 2020.

[7] T. Archer, "Test terraform modules in azure using terratest," Mar 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/developer/terraform/test-modules-using-terratest

[8] J. Saldaña, *The coding manual for qualitative researchers.* Langara College, 2022.

[9] A. Rahman and L. Williams, "Different kind of smells: Security smells in infrastructure as code scripts," *IEEE Security & Privacy*, vol. 19, no. 3, pp. 33–41, 2021.

[10] W. Chen, G. Wu, and J. Wei, "An approach to identifying error patterns for infrastructure as code," in *Proc. of the 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW'18)*, 2018, pp. 124–129.

[11] M. M. Hasan, F. A. Bhuiyan, and A. Rahman, "Testing practices for infrastructure as code," in *Proc. of the 1st ACM SIGSOFT International Workshop on Languages and Tools for Next-Generation Testing*, 2020, p. 7–12.