MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736

Prompt based complete Automated multi Cloud Service Management

[1]Pali Deepak Phani Krishna, [2]Dr.Sunitha Pachala,
[1]Student,Department of CSE, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeswaram, Guntur Dist, Andhra Pradesh, India, palideepak1549@gmail.com.
[2]Associate Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Green Fields, Vaddeswaram, Guntur Dist, Andhra Pradesh, India,
Email: drsunitha.pachala@kluniversity.in

Abstract:
This study discovered that the application of manual scripting method is a massive issue as far as managing multi-cloud architecture is concerned in the contemporary cloud.The proposed solution to these issues in this paper consists of a new intelligent automation solution by combining Terraform and Large Language Models.With the strategy, it becomes much easier to manage Infrastructure-as-Code across multiple clouds.On this platform, you can just need to create, query, and visualize cloud infrastructure using natural language instructions, which connects to major cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).It is supported in three key ways: Prompt and terraform-compatible infrastructure provisioning code generation; Terraform State Visualization plain-English table modeling of deployed infrastructure; and Multi-Cloud Resource Output Extraction easily available resource data extraction with simple Terraform installations.Research indicates that this technology will be able to automate end-to-end Infrastructure as a Code operations that can radically reduce the provisioning time.This time saving could be fifteen to twenty minutes or less than a minute depending on the complexity of the service.Besides its ease of use, the system emphasizes on security with the capabilities such as minimal-privilege execution and isolation of environments.This smart automation system will make the cloud easier to operate by non-expert users and act as an AI-based toolbase in the future of the DevOps environment [4].

Key words: Automation, Deployments, DevOps, Multi Cloud Providers, Software,Terraform, Largelanguage Model

Introduction

The increasing use of multi-cloud strategies by organizations has raised the need for automated infrastructure provisioning that is effective, consistent, and efficient [20].Infrastructure as Code (IaC) solutions such as Terraform may be a necessity to have a good understanding of resource management on platforms such as AWS, Azure, and Google Cloud.When dealing with distributed and complex infrastructure, most developers, system administrators and even DevOps engineers might find manually writing Terraform code, or updating scripts a tedious, error-prone and time-consuming process.This paper presents a mass-scale automated solution to the issue, through a custom-trained Large Language Model (LLM) [2].In order to construct or alter the Terraform code in accordance with user needs stated in plain English, the LLM's main goal is to comprehend these requirements [10].

Unlike other AI-based code generators, our LLM is trained on a handpicked collection of real-life Terraform scripts and modules stolen off of the official GitHub accounts of cloud providers such as AWS, Azure, and Google Cloud.The result is infrastructure code that is both correct and up-to-date, just what cloud providers need.As an example, users can create a virtual machine in AWS with a few keystrokes that can have access to S3 and a security group.In response to a request, the system checks with the trained LLM to produce provider-specific, syntactically sound Terraform code.This would enable anybody, not just professionals to create infrastructure without typing manually the HashCorp Configuration Language (HCL).Terraform generates code which is automatically stored and is run out of a configuration file (such as main.tf).By planning and executing the adjustments using Terraform CLI and capturing the results, the system can offer real-time information on the deployment's progress.Four best-practice-based transformations which the system provides during installation after smart analysis of the existing infrastructure are to activate logging, apply lifecycle rules, and specify access control.The system will also optimize the infrastructure by running the code numerous times with the adjustments made by the user.

Literature Survey

The work was done by B. Ray, S. Chakraborty, W. U. Ahmad, and K. Chang (2021)."Shared training on understanding and generating programs.This paper introduces a better sequence-to-sequence model of program-understanding and code-generation called PLBART.Python and Java code that is aligned with natural language is used by PLBART to learn the structural, syntactic, and stylistic features of code by denoising autoencoding.Experimental results show that PLBART is on par with or even better than state-of-the-art approaches on a variety of tasks, including code summarization, translation, software repair, and vulnerability discovery.This is used to develop coherent solutions to both generative and discriminative programming challenges by working reasonably well with minimal human annotations, which show high efficiency and flexibility.

The authors of the article by Odena, Nye, Bosma, Dohan, Jiang, Cai, Terry, Le, and Sutton changed their name to Austin et al. in 2021.Synthesis of programs for language models on a big scale.Program synthesis using massive language models, which can handle 137 billion parameters, is the focus of this study.MBPP and MathQA-Python are two significant measures when it comes to evaluating the accuracy of testing the models in creating Python code when given a natural language prompt via few-shot and fine-tuning strategies.The findings indicate that the performance of the model is much enhanced with increasing size.The model has 83.8 accuracy in MathQA-Python, and 59.6 in MBPP using few-shot prompts.Although the study examined the role of human input in mitigating model errors, it concluded that even with a large-scale model, it is impossible to achieve a precise prediction of the results of a program.This means that deep semantic comprehension is yet unexplored.Brown et al. (2020) discovered that language models are few-shot learners in their investigation.I will present GPT-3, a new language model that has been trained to solve a variety of natural language processing tasks, and has 175 billion parameters, and how it uses few-shot learning.In comparison to techniques relying on task-specific fine-tuning, GPT-3[9] can deliver task-specific outputs using few instances of training.The model is very effective in a variety of areas, such as machine translation, question answering, text-generation, and reasoning.While GPT-3 has accomplished a lot, it isn't perfect.As an example, it cannot cope with employment in the absence of training data and it cannot comprehend the natural language.Through prompt-based learning and massive pre-training as an interface to natural language tasks, presented intuitively, this work suggests a new way in which models can generalize and effectively address problems with previously unknown solutions.

Besides Dean and Bosma, Narang and Devlin (2022) are also a part of the study team.PaLM: Scaling language modeling with pathways.The project introduces a transformer model called PaLM, trained using the Pathways system by Google, having 540 billion parameters.PaLM exhibits a high level of few-shot learning across a broad spectrum of activities such as reasoning, operations related to code, language production and comprehension.It performs better than human on BIG-bench type multi-step reasoning questions and previous state-of-the-art models.The source code generation features and the high level of multilingualism of the model make it possible to automate chores, and increase the productivity of developers.Considering the fact that the performance of this model is non-linearly dependent on its size, the authors discuss the potential biases of the model, risks associated with memorization and ethical concerns of this model in detail.This information has been written by Dettmers et al. (2023).Quantizing LLMs efficiently and fine-tuning quantized LLMs (QLoRA).QLoRA is a method that can be used to refine large language models, including those with 65 billion parameters, with just a single 48GB GPU and a small amount of RAM, as described in this paper.Pages optimizers, Low-Rank Adaptation (LoRA), double quantization, and 4-bit Normal float quantization (NF4) are some of the methods it may be trained.Without compromising the model's accuracy, this strategy decreases memory needs.The finished model family, Guanaco, is as good as ChatGPT in terms of adhering to instructions.This study demonstrates that useful fine-tuning can be attained using minimal computational resources and explores the usefulness of existing chatbot benchmarks.The pre-trained model of the programming languages and natural languages used by Feng, Z., Guo, D., and Tang.The reason why CodeBERT[12] is a transformer architecture is that it can be trained on both programming language and plain English data.It implements a mixed goal based on bimodal (a combination of computer and natural languages) and unimodal data, which is through masked language modeling and used token recognition.CodeBERT is blazing fast when it comes to generating documentation and finding code.It has shown encouraging results in zero-shot settings and is able to perform well on a number of tasks involving natural language programming.A strong grasp of the semantic link between programming and common speech is shown by this paradigm.

The following people wrote in 2021: Zaremba, de Oliveira Pinto, Kaplan, Yuan, Tworek, Jun, and Chen.Assessing major language models via the use of code.The aim of this project is to enhance natural language program synthesis with the help of Codex, the language model based on GPT.It has been fine-tuned with the Python code, which could be located at GitHub.Codex is also benchmarked on the HumanEval benchmark to test its ability to solve a range of programming

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736

problems in Python.Codex is more accurate in solving problems than its predecessors with a maximum of 70.2 percent repeat sampling.To illustrate the excellent capability of Codex to generate practical code that executes human code, we can use the example of the commercial product GitHub Copilot.However, there are still challenges with large or long code, especially with more complicated bindings and logic operations.The authors do not merely dwell on the technical performance but the broader effects of using these models.They emphasize the need to look at the social consequences, correctness, and security of the large-scale AI code generation. Howard, M. According to (2022), there is a need to incorporate the software development lifecycle (SDLC) concepts into the infrastructure management, i.e. versioning and source control.Manual deployment is becoming less and less effective and more likely to fail as the infrastructure environment of the world is growing exponentially.You can set up resources, audits and deployments with a code-based solution of Terraform through Infrastructure as Code (IaC).The ability to automate infrastructure management enables it to keep up with the rigor and scalability of software engineering, and consequently enhance DevOps techniques in cloud-based operations by providing qualities that are applicable universally, such as consistency, traceability and reliability.The authors describe themselves in the article LoRA: Low-Rank Adaptation of Large Language Models published in 2021 by Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W.The strategy proposed by LoRA to fine-tune on a large scale is to inject the trainable low-rank matrices into each layer and freeze the major weights.This can cut the memory used by up to 10,000 without affecting memory performance, as well as cut the number of trainable parameters.LoRA can succeed, albeit not necessarily better, when fine-tuned with models like GPT-3, GPT-2, RoBERTa and DeBERTa.The success of LoRA might be observed in the experiments and evaluation of transformer rank deficits.The problem of releasing a PyTorch package is especially helpful to implement large models in small processors, which is beneficial to adapting them.Masolo wrote the article in the year 2023.InfraCopilot: Conversational Infrastructure-as-Code Editor.InfraCopilot, with its conversational interface, makes it easy to translate user-specified infrastructure requirements into fully-functioning Infrastructure-as-Code (IaC) solutions.The key aspects of it are the Discord bot that allows users to communicate with each other, an LLM-based intent parser, a Klotho-based architectural assembler, and a visualizer of feedback in the form of a graph.InfraCopilot enables IaC because it is accessible to all users, caters to users with different levels of technical ability, and allows for both low-level and high-level recurring improvements.It bridges the gap between the worlds of deployable infrastructure, which is structured and version controlled, and the power of simple language.Xiong, Zhou, Tu, Hayashi, Pang, and Nijkamp were included in their 2022 publication.CodeGen: Multi-Turn Open Large Language Model of Code Synthesis.CodeGen[10] is an open-source repository of publicly available, scaled, program synthesis-trained, large-parameter language models that can have as many as 16.1 billion parameters.The models can accept the input and output of code and plain English.CodeGen does not only perform well in benchmarks such as HumanEval, but also suggests the Multi-Turn Programming Benchmark (MTPB) that shows that interactions with multiple steps are better than synthesis on its own.In order to enhance the precision and velocity of high-level synthesis, the JAXFORMER training toolset promotes the modularization of complex computer code.

Among the authors of Pujar et al. (2023) are Buratti, Guo, Dupuis, Lewis, Suneja, Sood, Nalawade, Jones, Morari, and Puri; the list of authors is topped by Pujar himself.Granted Language Models in Automated YAML Code Generation in IT Tasks.The code of information technology automation, specifically the Ansible scripts, can be written in the YAML format with the aid of Ansible Wisdom, which is a transformer-based approach.According to the article under consideration, it is implied that.It automates and simplifies the translation of common language requirements and converting them to formal computer code.The system is much better in accuracy of tasks and in BLEU score as compared to the previous methods.As a result, it is possible to train domain-specific data, such YAML scripts, to beat larger, more generic models."Challenges of a DevOps Infrastructure. (Siebra, Lacerda, Cerqueira, Quintino, Florentin, da Silva, Santos, 2017).This paper gives a glance at the issues associated with adopting DevOps through Infrastructure as Code (IaC).The article gleans the most relevant information by comparing PowerShell DSC with Chef and Puppet and automating systems using them.The recommendations would involve standardizing the procedures and providing strong tool support along with pointing out the difficulties of increasing automation in various settings.Zhang, Pan., Zhang, Xing., and Sun, Y. published their article in 2025.Deployability Centric Infrastructure as Code Generation: LLM Based Iterative Framework.TheIaCGen framework use massive language models and iterative feedback loops to produce Infrastructure-as-Code (IaC) templates, with the goal of resolving the issue of prompt-to-code approaches' lack of deployability.The IaCGen team has a 98% success rate after testing 153 deployment scenarios.This demonstrates that the LLMs are sound reliability assurances. creating blueprints to architecture manufacturing processes through cyclic tests and improvements.

Methodology

prompt to multi provider service creation: The article provides the specification of an Infrastructure-as-Code (IaC) generation system that is responsive to user input and generates code with Large Language Models (LLMs).The methodology consists of mainly three stages:Multi-Cloud Terraform Code Execution and Deployment As-A-ServiceAutomated feedback loops are a characteristic of systems that propose infrastructure enhancements.To show how to fine-tune a Large Language Model (LLM) to translate natural language processing (NLP) instructions into Terraform configuration code, this step can be incorporated in the process of developing prompts in multi-cloud services.A dataset that was specifically designed for the LLM's domain was used for its training.thetf files that were retrieved[3] from open-source Terraform GitHub sources.The data was translated into pairs of prompt completions in the JSON Lines format.It used Terraform blocks as completions and the prompts were specified with natural language.

Special care was taken to:For example, you might start an EC2 instance and give it the name deepak. Prompts should maintain tag information, such as tags = {Name = "deepak" }. To improve generalizability, declare parameters differently.

The last dataset contained 25,000 pairs, 80 percent of which were training pairs, 10 percent validation pairs, and 10 percent test sets.A version of the Byte Pair Encoding (BPE) tokenizer was trained on the Terraform corpus with modifications before preprocessing or tokenizing [18].Details of the Token Generator:

- Vocabulary size: 30,000
- Special tokens: <s>, </s>, <pad>, <unk>
- Pre-tokenization: Whitespace-based
- Post-processing: Template-based for causal modeling

A dusting was performed a bit.To ensure the accuracy of the mapping we deleted comments, refrained from changing IDs and left field values intact.Since GPT-Neo 125M (EleutherAI) is a code-generating model, it was chosen as the fine-tuning model.

Work flow

The model produces Terraform code in the form of HCL by using tokenizer.pkl that tokenizes the prompt during inference [15].The outcome entails the generation of things such as dynamic naming of resources and creation of valid syntax.We adapted this model to our web and CLI chatbot interfaces to allow real-time IaCcreation.The pipeline below will automatically deliver infrastructure when creation of the code is completed via the Terraform command line interface.The Terraform configuration that was built is located in the main.tf file. The terraforminit script takes care of setting up the working directory, accepting the plugins delivered by providers and configuring the backends. Last but not least, the terraform verify script makes sure there are no semantic or grammatical mistakes in the setup.The Terraform planning tool creates an execution plan to showcase the alterations without actually applying them. Terraform implements the provisions of the Application section or configuration-dependent changes in the cloud infrastructure.

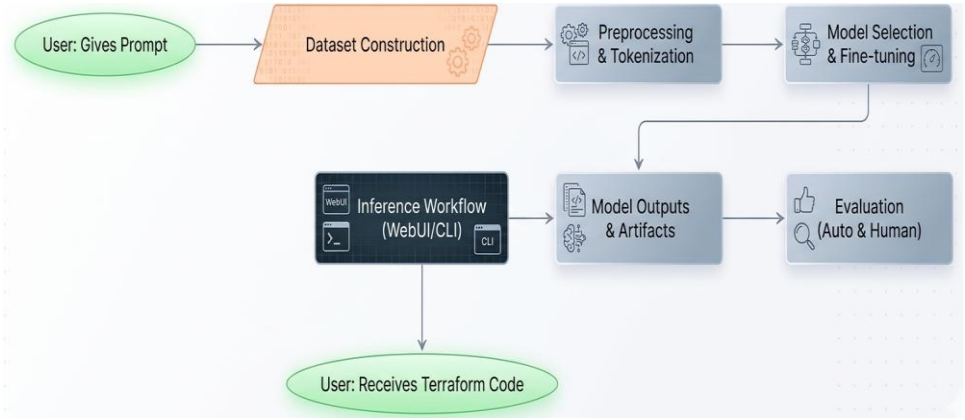
In this case, we apply the proposed improvements to the architecture base on the suggestions on LLM. The system will be operated in conjunction with the self-refining infrastructure loop.Triggering suggestion starts a suggestion cycle, after the initial deployment, with the generated configuration and prompt being recycled. The model assesses the infrastructure, along four lines: optimization (cost reduction, e.g.), security (authorization, e.g.), performance (resource tuning), and usability (tagging, readability, e.g.) [5].In order to generate four well-structured, implementable ideas, LLM-Based Suggestion Generation uses the same LLM. They are depicted through user interface buttons that can be clicked.After the user picks a recommendation, it gets displayed as a new prompt, the LLM changes main.tf, and the alterations are effected by re-executing the Terraform process (start → validate → plan → apply).

Iterative Optimization Loop

This is a loop in which one can make continuous improvements.One can develop an infrastructure cycle that builds itself up as new ideas are generated with each change that is made.With the help of a Network of Clouds[8]Our existing model is optimized with AWS Terraform settings, although you can add features unique to each provider and scale it to other providers with ease.uploading the tf files to the data set and changing the assessment criteria and prompt models.

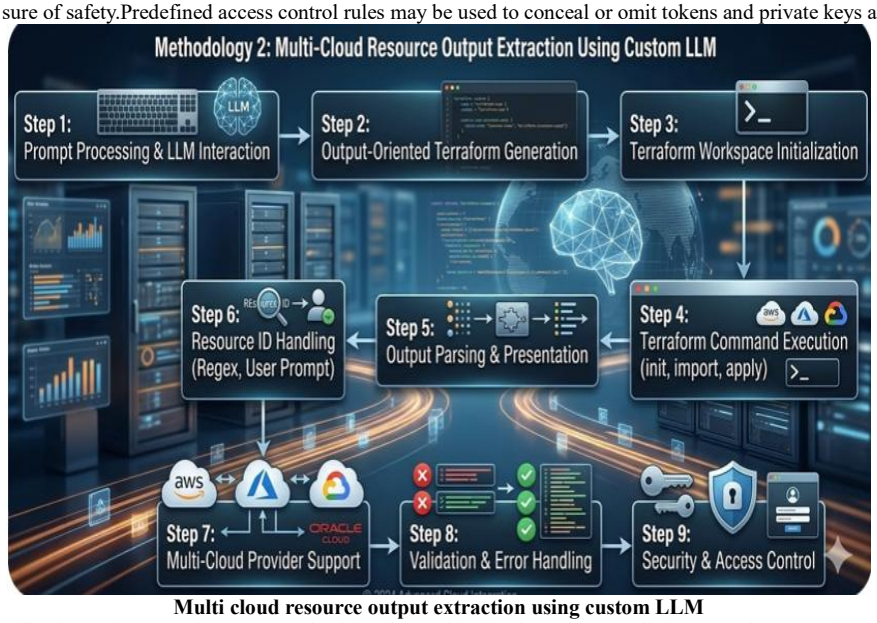
MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736

Multi-Cloud Resource Output Extraction Using Custom LLM:Using Terraform and a custom-trained Large Language Model (LLM), the technique details an automated procedure that can get live information about cloud resources (such as IP addresses, endpoints, and URLs) from many prominent cloud platforms including AWS, Azure, and GCP.Furthermore, it uses natural language prompts to produce Terraform output configurations that are particular to resources, all without the need for extra infrastructure.



Fiure.Architecture Prompt to Terraform code Generation (M1)

A query that could be typed by the user in Prompt Processing and LLM Translation is something like Get the IP address of my AWS EC2 instance with ID i-0abc123.The LLM takes the desired attribute (e.g., public IP, endpoint URL) and generates the Terraform configuration code, which contains a provider block specific to the cloud platform (AWS, Azure, or GCP), a resource block, which includes existing infrastructure and which imports it, and an output block, which extracts the desired field.In this case, we connect infrastructure questions to human will through code generation.The LLM has fine-tuned its output-oriented Terraform generation to: adhere to provider-specific Terraform conventions, refrain from building unsupported things (such data sources) until absolutely required, and give priority to retrieving outputs over providing them.Training utilizes a great variety of resource types and result-oriented queries to make sure that the code providers deliver the same code on a regular basis.To create a workspace where Terraform can be run, a workspace is created each time the command is executed.There is a main.tf file in this workspace that contains the generated code.With isolation, you can be certain that previous setups and state files will remain unaffected.The initial steps of the Terraform Execution Lifecycle are resource import, installing of a plugin and dependencies, and terraform init.When importing a Terraform resource, don't forget to specify the correct address (such as aws_instance.example) and authentic identification (such an EC2 ID) in case the destination resource has already been created.Instead, the application of Terraform manages configuration registration and evaluation of output blocks: apply.After completing the application phase, the user can use the Output Retrieval and Display section to view the results of his/her queries.This makes it unnecessary to go through cloud provider dashboards one by one to get the information you need, such the virtual machine's public IP address, storage endpoint, or API URL.Since the assets and improvements are identified by locating them and adding promptsResource IDs (such as i-0abc123) to the generated Terraform to help understand what was created. The interaction with the system lets people request missing identifiers interactively which are then either extracted or inferred with regular expressions or semantic parsers.The LLM can detect AWS, Azure and GCP platform-specific syntax as part of its Multi-Cloud Provider Support.The syntax of AWS, Azure, and GCP is aws followed by the right region and access key, azurerm followed by the right client id, tenant id and subscription id, and a provider of the type of Google followed by the correct project id, region and service account of GCP.The context of the prompt and the user's input are used to dynamically pick the provider.All of the Execution Robustness and Validation process (init, import, apply) elements include validation tests of Terraform syntax and provider compatibility. issue messages give the user detailed diagnostic instructions in case of an error.The procedure aims to ensure that it does not damage the existing infrastructure and is solely aimed at extracting data.The Terraform backend uses Security and Credential Management to use secure credentials.It is possible to obtain these qualifications with the help of GCP, Azure, or AWS.Always put output on read-only to be sure of safety.Predefined access control rules may be used to conceal or omit tokens and private keys among other sensitive outputs.

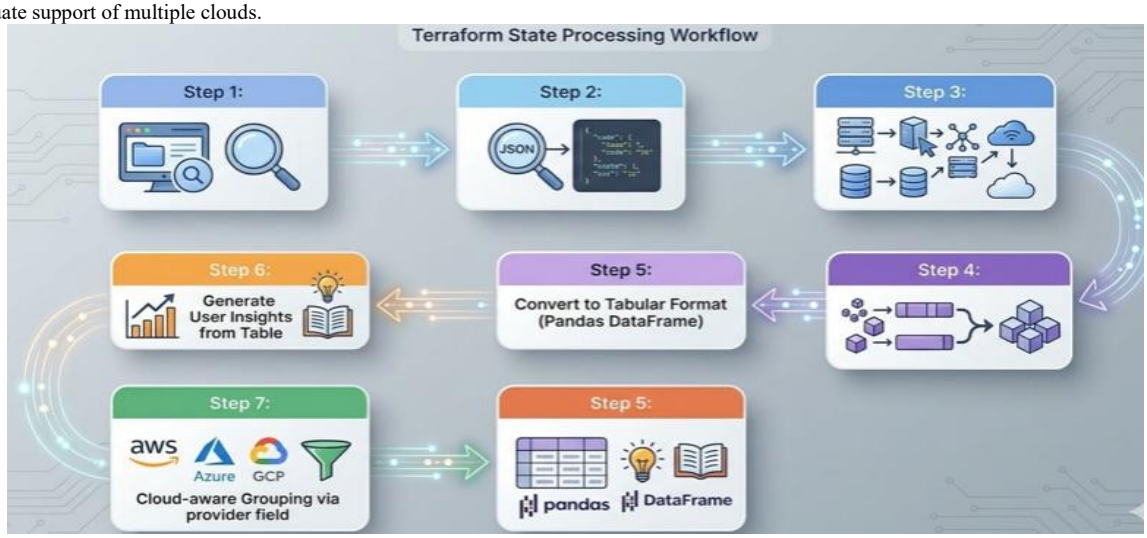


Multi cloud resource output extraction using custom LLM

Multi-Cloud Infrastructure Verification Using Terraform State: Checking the Multi-Cloud Service Installation Based on Terraform State Files.This software can measure the progress of cloud infrastructure on AWS, Azure, and GCP without having access to the control panel of any cloud provider; it is non-invasive and cloud-agnostic.Instead it relies on a type of surety system, the Terraform state file (terraform.tfstate) to demonstrate the services supplied, recognize them, and validate them with inputs entered in plain language.The Terraform state file can be retrieved and located with the help of Terraform.This file keeps track of the status of all deployed infrastructure.The system will automatically open the tfstate file at a known place once a provisioning event (like terraform apply) happens, for example, terraform workspace/terraform.tfstate.This file is used as the gold standard to verify that the necessary services have been supplied.Systematic Validation Program LoadingThe state file is JSON compliant with regard to structure.It is programmatically loaded in memory by the JSON parsing libraries.To ensure that the files won't be confused or cause errors in the future, we check their presence and the format of the files.In the Resource Block Extraction, all the deployed infrastructure elements are stored in the resources key of the state file.Two kinds of data are retrieved on each resource.Metadata contains details about the name of the resource, its type, the mode of provisioning (managed or data) and the cloud provider (Google, Azure or AWS).Attributes that contain actual configuration information of deployed instances such as URLs, IP addresses, instance IDs and ARNs also exist.To simplify the attribute structures, the complex

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736

attribute structures are simplified using a recursive flattening algorithm into key-value pairs.Look at the following example: the value of the network interface.network_interface.public_ip is now the default value of public-ip.Structured Normalization validation happens on numerous occasions on each resource block.Flattened attributes are combined with metadata and give a format that is arranged and can be analyzed.This proves the usefulness of the resource provided it was created and used with some quantifiable objectives in mind.Each of the combined resource cases will be contained in one dataset.The first point in the list of Conversion into Tabular Acknowledgement View is to build a DataFrame in Pandas.This is only one of the many functions that can be performed by exporting to CSV or Excel to report.You may also sort by important variables such as the creation date or location, filter by cloud provider, resource type or deployment status and much more.The results can be in a tabular form which is easy to digest (fancy_grid or an alternative) using the tabulate library to enable the user to quickly determine the status of deployment of resources.Users can check what resources have been generated successfully during the Confirmation and Insight Generation process by reading the outcome table.Obtain the corresponding cloud provider to each resource and cross-check their names, endpoints, and IP addresses, which are all important configuration variables.With multi-cloud configurations, effective post-deployment checks can be conducted without requiring a cloud dashboard.This method can be applied to any provider (not cloud-specific) because terraform.tfstate is constantly collecting resources in AWS, Azure, and GCP [19].Every resource block has a provider field that may be used to automatically recognize cloud platforms.This will ensure consistency in service checks as well as adequate support of multiple clouds.



Visualization of Terraform State Files[7]

It is an image of the Terraform infrastructure which accepts the.tfstate file and converts it into a tabular form of the deployed infrastructure.Metadata can be easily filtered, sorted and exported by viewers.Provides a more user-friendly way of viewing raw state files or command line output.Converts complex infrastructure data into human-readable form easily and quickly.

Errors Encountered During Development: The use of these methods brought up a number of practical problems.It was however always a challenge to come up with perfect Terraform code despite having trained the LLM to generate Terraform setups on an individual basis through prompting since the generated output usually necessitated multiple general revisions.The error-free production-ready.tf file was hard to obtain and could not withstand intricate settings.Terraform apply may fail due to minor errors, like provider settings that are not correct, syntax errors, dismatched resource attributes, etc.In importing resources into Terraform, some possible problems may happen leading to execution failure.These are conflict of names, lack of backing import paths, and lack of available identifiers.It has been a tedious and error-prone procedure to consistently use these IDs across clouds.State file management: The logic of recursive programming was required to handle the highly hierarchical formats of state files used by Terraform, and transform them into a table that can be read.Flattening was further complicated where the edge had no resources attached to it, was of defaulted nature or where there was no resource attached to the edge. mechanism.Following the mistakes, further study and testing were conducted to fine-tune the actual application.The following are methods that we explored prior to arriving at the current design:Firstly, we suggested using web scraping to get fragments of the codes found in the Terraform Registry.However, it was susceptible, unsustainable and reliant on HTML structures, which can easily be changed.It could also not be tailored to the individual user preferences.Introducing LLM and Terraform GitLab Together: The option to directly work with data provided by the repositories of Terraform modules by the suppliers in the GitHub was also explored.One possibility is to build a database of raw.tf files, either on-premises or on-cloud, and then train the LLM over it.Although it may have worked in theory, it needed a lot of preprocessing and couldn't generate code in real time.Public LLM APIs such as as OpenAI and Cohere were also tested to convert prompts to code.These models however were a complete failure in regards to multi-cloud design logic and Terraform syntax.And not just that, but it could be difficult to fine-tune, and the Terraform materials it generated tended to be misleading or even hallucinatory.Recommended strategy: The final option was to train a custom LLM in the use of the documentation, the cloud-specific resources, and Terraform modules which constitute the Terraform knowledgebase.Preprocessing was done on the dataset.As a result of this, it became easier to provide the correct results and respond to user feedback as per Terraform requirements.There are still some limitations due to the application's design and Terraform's inherent restrictions, even if this work's approach has enhanced the multi-cloud automation process.Lack of Real-Time Data Access:Terraform is declarative, thus it can't access data that changes in real-time, like use or operation statistics.It merely reflects the latest state, which was implemented.Terraform Import is not universal in supporting all types of resources.Certain resources are complex or hierarchical and have ways around or partial imports.By the way, you can not view the end result prior to executing the configuration in Terraform yet.Delays or the need to build fake resources could result from this restriction.Resource not found and invalid identification are both examples of ambiguous error messages that in most cases may require human intervention to debug and analyze.In the case of previously undocumented or unknown services, especially, the LLM Output Accuracy large language model can hallucinate[6][12] features that do not exist or miss the intent of the user, or it can generate redundant or missing elements of this critical base.The state file and the entire visualization process are all based on Terraform.Wrong or incomplete infrastructure views can be generated when the file is corrupt, out-of-date or partially written.Unique Cloud-Specific Variability: Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform (GCP) differ in their resource naming conventions and organizational structures.It would require a lot of adjustments and corrections to consider all the possible variations.Data Leakage and Security Suspensions It might lead to the release of data that shows sensitive resource information (IDs, IPs and passwords) due to inadequate process management.This can't happen without stringent security measures and controlled access.This project takes a giant leap forward in the effort to streamline infrastructure management across several clouds by combining Terraform automation with Large Language Models (LLM).It demonstrates how natural language interfaces can accelerate infrastructure functionality on AWS, Azure, and GCP, which further can empower the connection between individuals and Infrastructure as Code (IaC), decrease the technical barrier, and so on.As a step in the process,Start, import, and apply Terraform; Apply the LLM in an output-oriented way to minimise and synthesise the main.tf file; Find a message represented as spoken words Putting the attribute of the desired output (such as IP address, resource name, or endpoint URL) into the limelight.To ensure it is more interactive and reliable, the system can prompt the user to provide an explanation or regular expressions to derive resource IDs out of the prompt in cases where it is not clear.In addition, it can be used with a number of clouds, which means that it can adapt to different forms of login and provider-specific options in real-time.Terraform is based on the principle of minimal privilege because its backend uses IAM roles and service accounts with limited or no access.This safeguards against the unintentional replenishment or withdrawal of resources.Preexisting infrastructure secure inquiry.It may be necessary to temporarily disable or change some security settings to less restrictive settings so that Terraform can be able to support certain imports or settings of resources.To reduce extreme restrictions to output access, such as, you might need to disable them.This is to ensure that the system is not compromised by executing it carefully by monitoring and recuperating these parameters back to their pre-execution settings.As little and limited output is provided to the user as opposed to the so-called cacophony of full Terraform logs or CLI responses.This would lead to reduced mental load on the system and faster decision making which consequently makes it easy to use and efficient.We propose the following upgrades to enhance the system to be more intelligent, enterprise-ready, and resilient.Integration with APIs of services like

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736

GCP Operations Suite, AWS CloudWatch, or Azure Monitor may monitor the rate of system components like memory, storage, and CPU in real-time.Customers can monitor the health and efficiency of their resources live, along with configuration data, improving the capabilities of the inquiries that are static.In subsequent system iterations, the LLM-generated Terraform code might be continuously enhanced via RLHF [13].In doing so, a superior model could be developed through this adaptive learning.This model would be in a position to understand the infrastructure patterns, user intent, and cloud-specific best practices, including ensuring accurate and reliable delivery.System resilience can be significantly enhanced by adding a validation layer, which is capable of comprehending user instructions.Incorrect, missing, or ambiguous prompts would be detected by this component, and it would then provide solutions like auto-completion or suggestions for corrections.To reduce the number of failures and enhance the quality of interaction, such an approach becomes extremely important to non-technical users.The infrastructure code that has been developed by Terraform will be interacting directly with cloud environments, so it is necessary to have automated security and compliance checks.To keep infrastructure secure and within the standards of governance, such technologies as as Open Policy Agent (OPA), Checkov, or tfsec can be utilized to identify open ports, wrongly configured IAM roles, unencrypted resources, etc.The error messages can be slightly obscure and confusing to those not familiar with Terraform.The machine can check the most probable causes of the system failures, recommend a solution, or simply implement the repair based on the recommendation by changing the mistakes into diagnostic.This will address one of the key usability problems, as well as make the system more self-healing.A method of making infrastructure more visible is to offer a GUI that identifies the connections and linkages among resources.Users can interact with their infrastructures using topology maps with the help of technologies such as Terraform graph and JavaScript visualizations.This would be an addition to the main querying logic, but would greatly improve the user experience.Drift detection techniques can be added in order to maintain a watch on the deviations between the real condition of the infrastructure and the desired one.To render this attempt even more useful, it would be marvelous to have the option of secure deletion of the resources with preview and confirmation measures.Although the current emphasis is on query capabilities, it would be far more beneficial to utilize it for bigger infrastructure lifecycle management tasks.

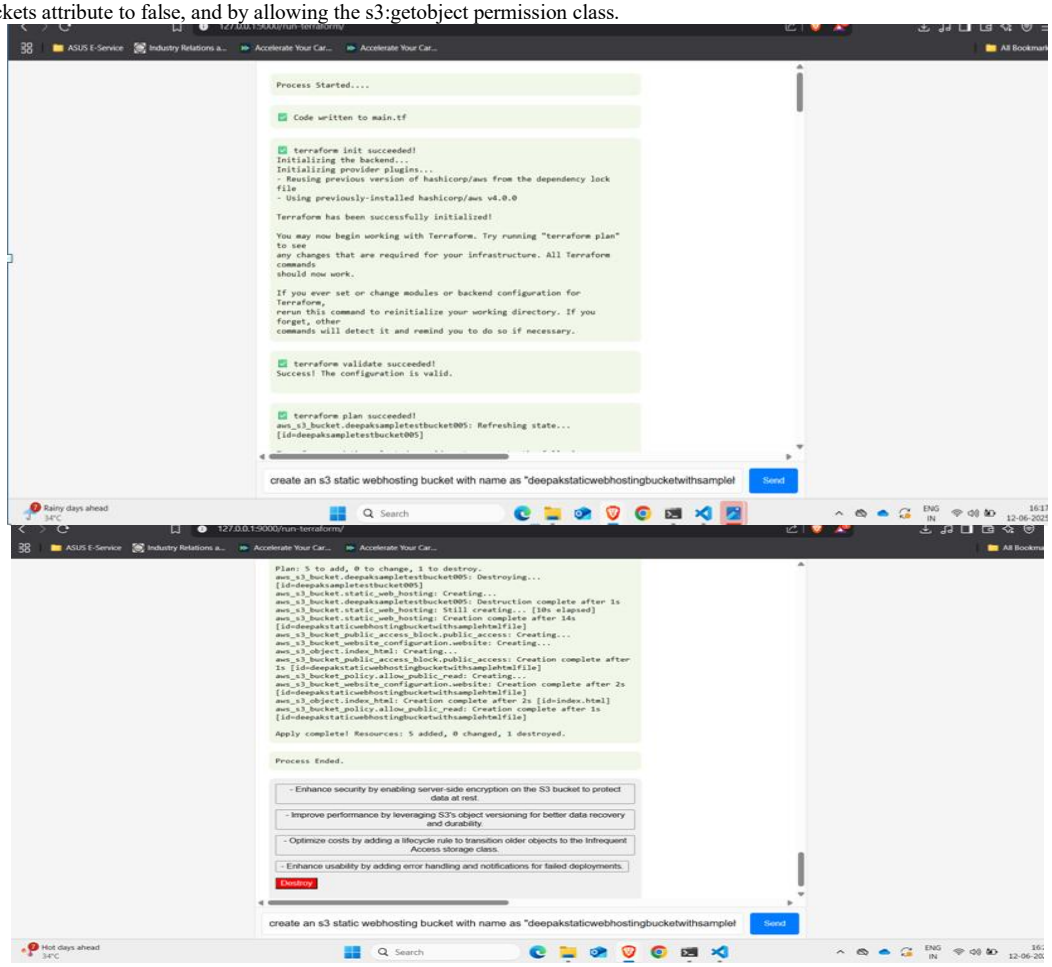
Discussion

This thesis attempted to automate multicloud infrastructure maintenance on a timely basis with the assistance of Terraform and Large Language Models (LLMs).Terraform integration with the natural language interfaces can be of great help to users and make the DevOps operations less complicated in terms of learning.With the help of three basic techniques introduced in this paper, highly useful automated processes were created with the Terraform and time-driven Large Language Models (LLMs).The duration of these processes is normally fifteen to twenty minutes.The time of development is directly related to the complexity of the service.Services using one S3 bucket, like a virtual machine service, only take less than 10 seconds.Less than one minute with highly-connected services (e.g., a virtual private cloud (VPC) with subnets, security groups, elastic computing resources (EC2), and a load balancer).Prompt-to-Infrastructure Code Generation method enables the user to build infrastructure by making simple English statements about it.Since the query is translated into Terraform code by the LLM, AwareDeployment is pulled out.Makes itself the gold standard in the provider set-up, flexible configuration and hand written manuals.Compared to the tedious manual system, simple installations only require a few seconds to set up and even very complicated multi-service designs can be brought online in under a minute.Multi-Cloud Resource Output Extraction allows users to query the cloud resources with natural sounding queries to retrieve cloud resource data in place of searching by IP address, URL, or storage endpoint.Terraform output blocks are dynamically generated to allow the system to access real data on AWS, Azure, and GCP.Accelerates the retrieval of correct infrastructure data, when compared to analyzing cloud consoles or making custom queries.

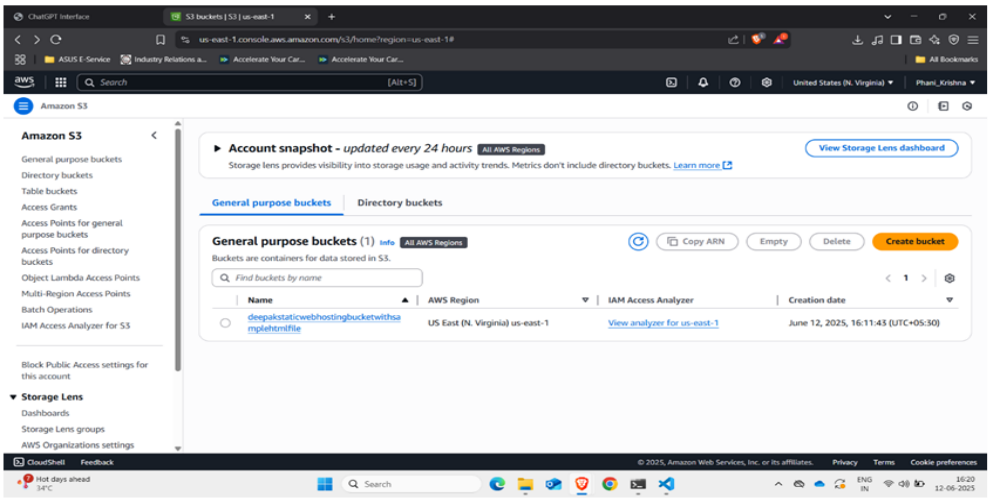
.Results

Prompt:Enter the name and environment in the tags, i.e., Static site bucket and Production.Secondly, make an Amazon Simple Storage Service (S3) bucket called deepakstaticwebhostingbucketwithsamplehtmlfile that will store your static web files

The HTML file must have the following properties added to it prior to uploading it to S3: key = index.html and source = C:Users/palid/Desktop/pr-service/temporary project/index.html and a content type of text/html This will permit the hosting of the static web.To prevent unauthorized access to the bucket, you can forbid it by deleting the public_access_block, creating a new resource with the name aws_s3_bucket_public_access_block, and keeping the restrict_public_buckets attribute to false, and by allowing the s3:getobject permission class.

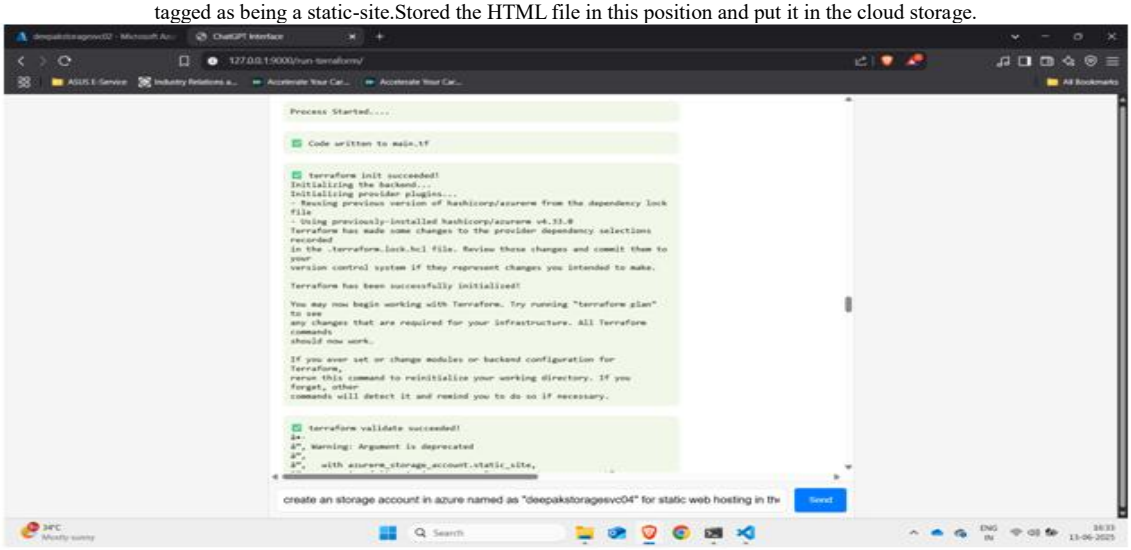


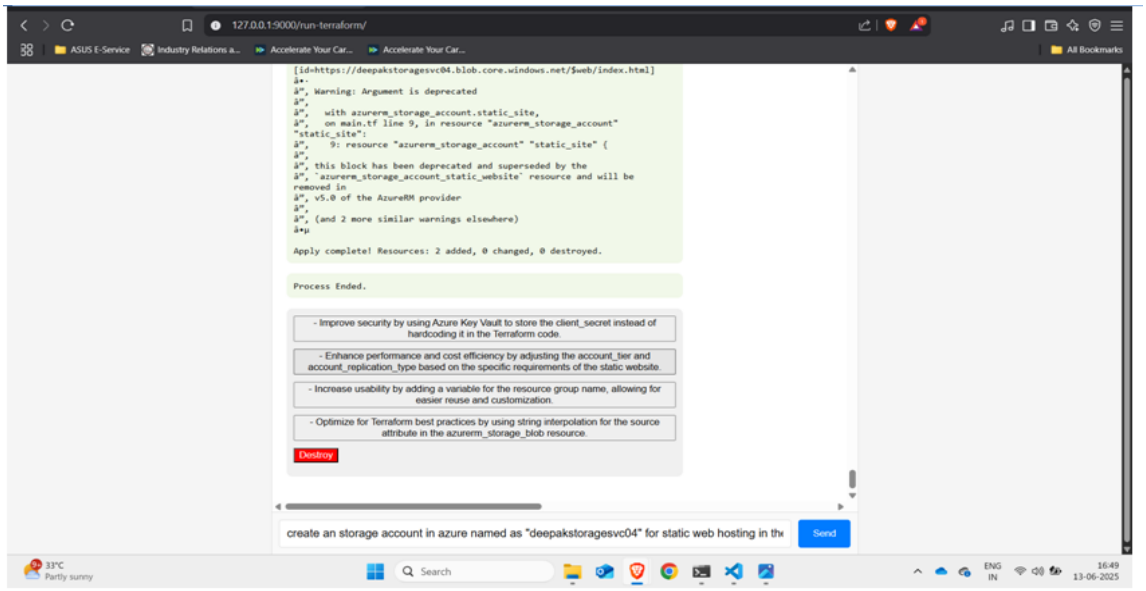
MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736



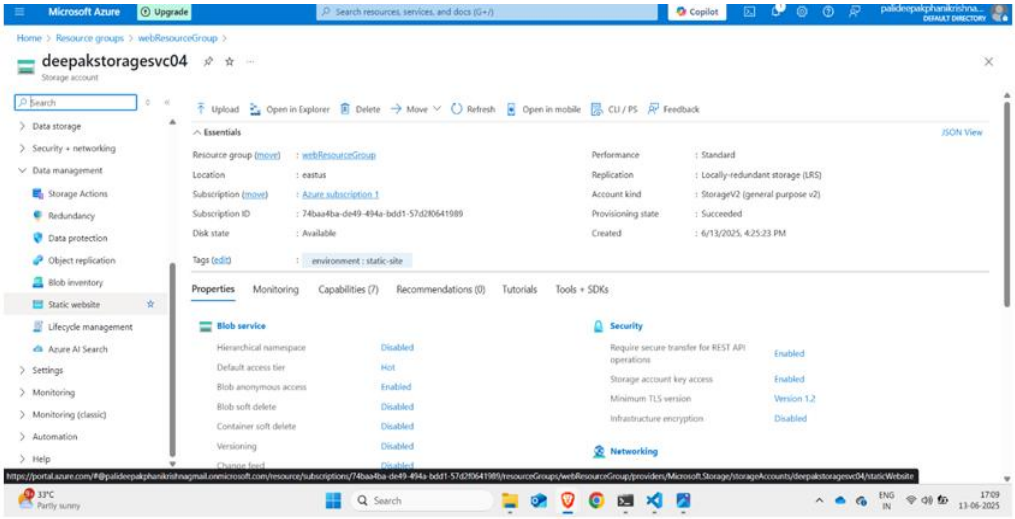
In Azure:Static Web Hosting:

Prompt:The new Azure storage service, which was called deepakstoragesvc04, was created in an existing Resource Group in the US East Region.Statics web hosting is now available on that resource.You have set index document in that resource as index.html and error 404 document as 404.html.The environment is tagged as being a static-site.Stored the HTML file in this position and put it in the cloud storage.

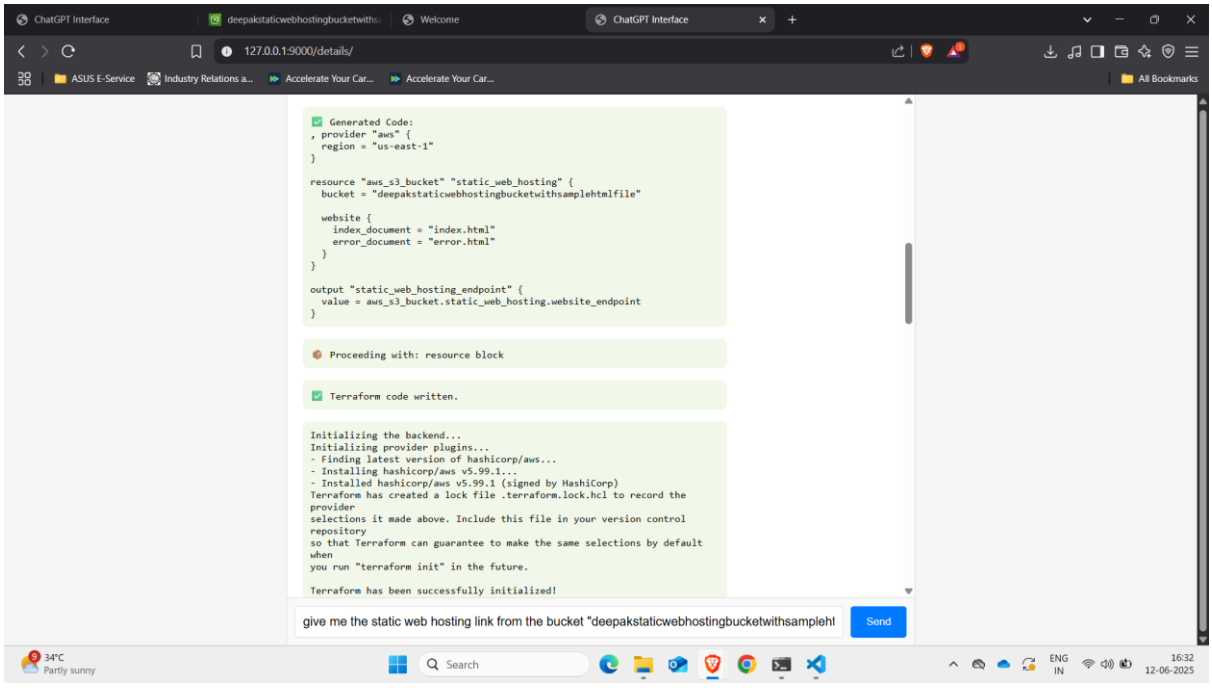




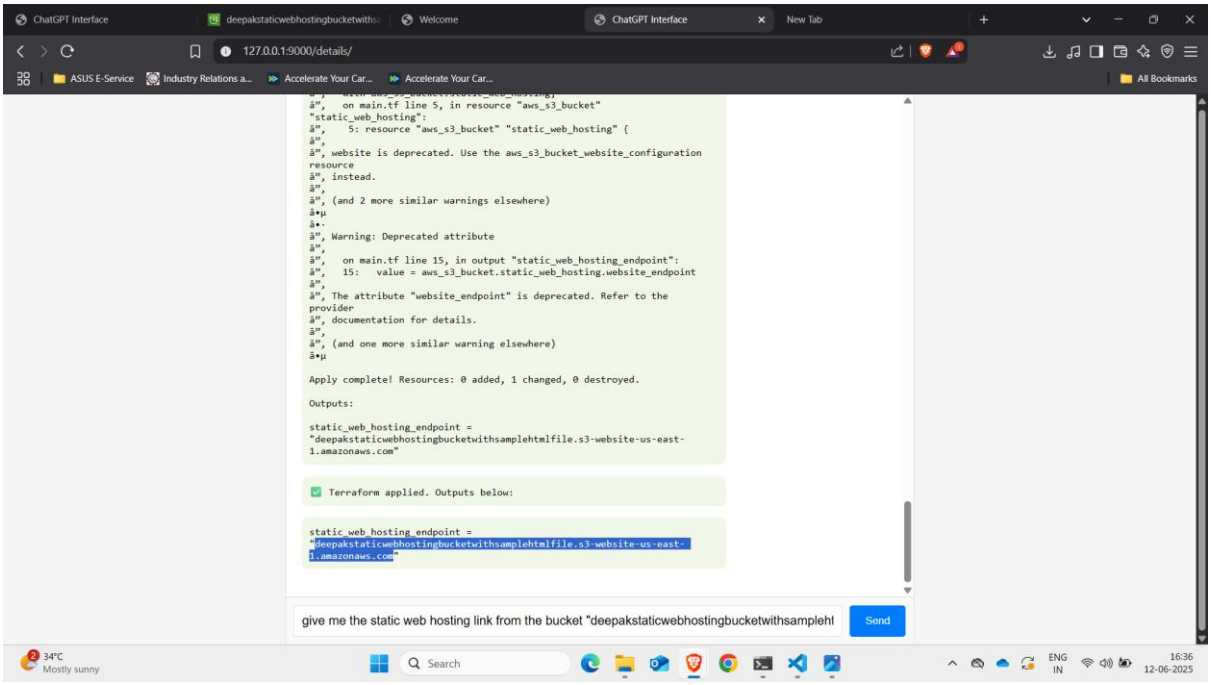
MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736



Details Extraction :
Prompt:Please share the URL to the "deepakstaticwebhostingbucketwithsamplehtmlfile" static web hosting account with me.



MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736



Disadvantages:

1. Absence of State Management: Since it does not maintain or refer to a permanent Terraform state file, the system is oblivious to the previously given resources.Not being able to distinguish between the current infrastructure and the future needs, it indulges in unproductive and even perilous activities.

2. Destructive Re-provisioning Behavior: Each run is equivalent to a deployment cycle.The system tends to rewrite the old resources and recreate them rather than making changes in a more incremental manner.This method does not fit well in environments which require consistency and reliability.

3. Risk of Data Loss: Cloud storage buckets, databases and volumes are often susceptible to decommissioning and re-provisioning, which significantly contributes to the possibility of losing data.This system does not give protection to stateful data a priority.

4. Lack of Idempotency: There is no system that provides idempent operations.Running the same input repeatedly also contravenes a key principle of infrastructure automation as it raises the chances of unintended changes or redefining resources.

5. No Drift Detection Capability: In the case of users manually editing their cloud configuration, the system lacks the configuration drift that would occur.Inconsistencies cannot be monitored or fixed unless the present infrastructure status is compared to a known baseline..

6. Inefficient Resource Utilization: The deletion and recreation of infrastructure during the deletion and recreation of infrastructure often wastes cloud resources, increases API requests and extends provisioning times.This inefficiency can also result in increased operating costs.

7. Limited Scalability: Complex or large-scale infrastructures are not good fits for this method.The amount of time, labor and possibility of error in total re-provisioning increases in direct proportion to the volume of resources.

8. Weak Fault Tolerance and Recovery: The system lacks the required procedures to proceed or rectify the deployment in the event of half failures when running is going on.The entire provisioning process is rather reinitiated, which brings about redundancy and instability.

9. Absence of Version Control and Change Tracking: The system does not store any history of past infrastructure changes.Due to this, changes are difficult to audit, version information is difficult to keep track of and returning the system to past stable configurations is challenging.

10. Security and Configuration Instability: Inconsistent security settings, such as changes to access restrictions, credentials, and policies, might result from repeatedly recreating resources.Temporary vulnerabilities or misconfigurations may be introduced as a result of this.

11. Dependency Management Issues: Complex cloud infrastructures are generally characterized by interdependent resources.Failure or inconsistency can manifest when the system methodology interrupts such dependencies when re-provisioning.

12. Reliability Concerns with AI-Generated Infrastructure Code: Careless inconsistencies between executions can occur because of the dynamically generated infrastructure specifications.Such fluctuations might cause unpredictability in behavior and deployment reliability in the absence of a steady reference state.

Future Enhancements of the Proposed System:

1. Integration of Remote State Management

The need to use remote backends to manage state centrally, such as cloud object storage with locking mechanisms, is one of the key improvements.In this way, the system might be able to track down resources that were already provided to it and perform upgrades safely, instead of being forced to reload everything at once.

2. Incremental Deployment with Change Detection

The system can be extended to enable a plan-and-apply workflow; in such a workflow, the proposed changes to the infrastructure are reviewed and depicted as a delta and only after that the changes are implemented.This will enhance efficiency and help to avoid squandering resources through needless modifications and changes.

3. Data Preservation Mechanisms

Safeguards like these should be included into future versions to deal with the possibility of data loss:

• Policies for the lifespan to avoid deletion by accident

• Backup/restore integration to support the essential resources.This is how stateful services like storage and databases are ensured to be operational.

4. Idempotent Execution Framework

Recurring executions can be safe and predictable by enhancing the system to ensure idempotency.Processing the same input over and over again may cause unforeseen changes hence it is significant to retain setups.

5. Drift Detection and Self-Healing Capability

To improve the system, the addition of drift detection to identify external or human modifications to infrastructure is used.This could be used together to enable a self healing system to automatically rectify errors and ensure nothing goes wrong.

6. Fault-Tolerant Execution and Recovery

Checkpointing and partial failure recovery should be included in future improvements.To enhance their rate of execution and dependability, the system must resume at the moment of failure instead of initiating the entire process once it fails.

7. Versioning and Audit Logging

It would be possible with the implementation of a system of execution logs and infrastructure version control:

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 2, 2026, Pages 728-736

• monitoring of changes with time.Restore to the previous good settings.Improved accountability and debugging.This is important in the validation of research and its application in the real world..

8. Security Policy Stabilization

The system can have predefined identity and access management templates to implement consistent security configurations.This makes it impossible to create security holes that had not been expected due to recreation of the resources.

9. Dependency-Aware Resource Provisioning

Less failures in complicated structures can be achieved with dependency mapping which enhances the knowledge of the system of the relationship of resources and optimization of provisioning sequence execution.

10. Validation Layer for AI-Generated Code

Terraform configurations developed by AI might also undergo an extra check and test to be more trustworthy.This can involve syntax validation, policy checking, and simulations to minimize unforeseen effects, prior to actual deployment.

11. Cost Optimization and Monitoring Integration

Potential features of future editions include estimation of costs before to deployment, and monitoring after provisioning.This information would enable customers to have a better view of their spending on the cloud and make informed decisions.

12. Multi-Cloud and Hybrid Cloud Support

An option that would make the system more useful would be to support a large number of cloud providers simultaneously.In this manner, customers would have a single interface to install and operate infrastructure across the environments.

Conclusion:

This project transforms the management of the cloud infrastructure by integrating the advantages of Terraform based automation with the advantages of Large Language Models (LLMs).This technology is enabling users to take action against multi-clouds (AWS, Azure, GCP), thereby reducing the technical barrier to designing and operating Infrastructure-as-Code (IaC) by means of natural language prompts.Three key approaches have been created and evaluated:Users can now deploy services in seconds with intelligently written Terraform scripts using LLM with prompt-to-IaC features.With multi-cloud output extraction, there is no longer a necessity to use a manual navigation and use the dashboard to obtain the real-time resource information.Terraform State Visualization might make the deployed infrastructure more readable, accessible and transparent by providing a tabular presentation of the infrastructure deployed.The use of existing resources, the creation of operational Terraform, and access to specific data on the cloud were all signs of the efficiency of the system.The timeframes ranged between fifteen and twenty minutes (with manual provisioning) to under a minute depending on the complexity of the service.Although the solution encountered issues like hallucinations with the LLM models, a limitation to the import functions of Terraform, and the inability to operate the state files, the end result was user-friendly, extensible, and stable.Other security controls were also taken into consideration like isolating execution contexts and application of least privilege principles.Finally, this preempts the AI-driven DevOps tools[14] and proves how verbal interaction is being moved towards infrastructure and efficiency automation and how all types of users, including cloud engineers and non-experts, can participate in the processes and functions through AI-Generated DevOps.

References

[1] Bishnoi, P. K., Kumar, D., &Bhanti, P. (2024). Terraform Framework Development for Multi-Cloud Docker Migration. Journal of ElectricalSystems, 20-02s, 1416–1426. https://journal.esrgroups.org/jes/article/view/8501.

[2] Chanus, T., & Aubertin, M. (2023). LLM and Infrastructure as Code Use Case. arXiv preprint arXiv:2309.01456. https://doi.org/10.48550/arXiv.2309.01456.

[3]NeurIPS. (2024). IaC-Eval: A Code Generation Benchmark for Cloud Infrastructure. Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS 2024). https://neurips.cc/virtual/2024/poster/97835.

[4] Parthasarathy, K., Vaidhyanathan, K., Dhar, R., Krishnamachari, V., Muhammed, B., Kakran, A., Akshathala, S., Arun, S., Dubey, S., Veerubhotla, M., & Karan, A. (2025). Engineering LLM Powered Multi agent Framework for Autonomous CloudOps. arXiv preprint arXiv:2501.08243. https://doi.org/10.48550/arXiv.2501.08243

[5] Jin, Y., Yang, Z., Liu, J., & Xu, X. (2025). Anomaly Detection and Early Warning Mechanism for Intelligent Monitoring Systems in Multi-Cloud Environments Based on LLM. Proceedings of the 2025 5th International Symposium on Computer Technology and Information Science (ISCTIS 2025). arXiv preprint arXiv:2506.07407. https://doi.org/10.48550/arXiv.2506.07407.

[6] PromptLayer. (2024). Can AI Generate Perfect Infrastructure Code? PromptLayer Research Papers. https://www.promptlayer.com/research-papers/can-ai-generate-perfect-infrastructure-code.

[7] CloudCycle. (2024). Visualization Frameworks in TOSCA-Based Deployments. CloudCycle Documentation. https://arxiv.org/html/2506.05623v1.

[8] SeaClouds. (2014). Adaptive Multi-Cloud Orchestration with TOSCA. CLEI Electronic Journal, 18(1), Article 1. https://riuma.uma.es/xmlui/bitstream/handle/10630/12561/SeaClouds-ECSA16.pdf?sequence=1&isAllowed=y.

[9] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). Training language models to follow instructions. NeurIPS 2022. https://arxiv.org/abs/2203.02155.

[10] Nijkamp, E., Bosc, T., Kanade, V., Meade, R., Wu, H., Dong, X., Zha, S., Ren, T., & Sabharwal, A. (2022). CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv preprint arXiv:2203.13474. https://arxiv.org/abs/2203.13474.

[11] Bonde, O. (2022). Generating Terraform Configuration Files with Large Language Models (Master's thesis, KTH Royal Institute of Technology). https://www.diva-portal.org/smash/get/diva2:1692943/FULLTEXT01.pdf.

[12] Srivatsa, A., Kumar, P., & Jain, R. (2024). Detecting Linguistic Inconsistencies Between Code and NL Descriptions. arXiv preprint arXiv:2404.00227. https://arxiv.org/abs/2404.00227.

[13] Oosterhout, J., Smith, L., Patel, A., Nguyen, Q., & Zhao, Y. (2023). Reinforcement Learning from Human Feedback for Code Models. Journal of Artificial Intelligence Research, 76(2), 1104–1128. https://doi.org/10.5555/rlhf4code.2023.1104

[14] LM-CloudComplete. (2024). Leveraging Cloud Infrastructure for Efficient LLM Code Completion. arXiv preprint arXiv:2405.06661. https://arxiv.org/abs/2405.06661

[15] Touvron, H., Lavril, T., Izacard, G., Martinet, X., et al. (2023). Code Llama: Open Foundation Models for Code Generation. arXiv preprint arXiv:2308.12950. https://arxiv.org/abs/2308.12950.

[16] Anthropic. (2024). Claude 3 Model Family: Opus, Sonnet, Haiku. Model Card. Anthropic Documentation. https://docs.anthropic.com/en/docs/models-overview.

[17] Hu, Z., et al. (2024). RTLLM: Benchmarking Code Generation with Transformer LLMs. Proceedings of the 29th Asia and South Pacific Design Automation Conference (ASPDAC 2024).

[18] Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. Physica D: Nonlinear Phenomena, 404, 132306. https://doi.org/10.1016/j.physd.2019.132306

[19] ISO/IEC 22123-1. (2023). Information technology — Cloud computing — Part 1: Vocabulary for Multi Cloud Architecture and Deployment. International Organization for Standardization. https://standards.iteh.ai/catalog/standards/iso/83fb23a3-9c08-4835-999e-40d2338c60b5/iso-iec-22123-1-2023

[20] Mulder, J. (2023). Multi Cloud Strategy for Cloud Architects. Packt Publishing.