TERRAFORM QUESTIONS

Did You Know?

- Terraform is written in the GO programming language.
- HCL (HashiCorp Configuration Language) is used as a description language in Terraform.
- Terraform was introduced to portray a piece of code as a complete infrastructure

1. What is Terraform?

- Terraform is an infrastructure as code tool that allows you to specify cloud and on-premises resources in human-readable configuration files that can be versioned, reused, and shared.
- After that, you can utilize a standardized procedure to provide and manage all of your infrastructures throughout their lifespan. Terraform can manage both low-level components like compute, storage, and networking resources as well as high-level components like DNS records and SaaS functionality.

2. What do you mean Terraform init?

- Terraform initializes the code with the terraform init command. The working directory for the Terraform configuration files is created with this command. It is acceptable to execute this command many times.

The init command can be used for:

- Plugin Installation
- Child Module Installation
- The backend is being set up.

3. Who are Terraform's main competitors?

The main competitors are:

- Ansible
- Packer
- Cloud Foundry
- Kubernetes

4. What is a Terraform provider?

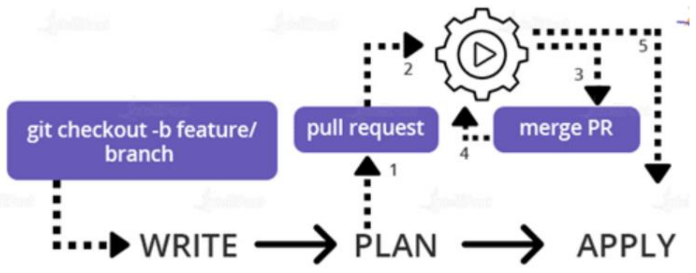
Terraform is a software application for controlling and informing infrastructure resources such as physical computers, virtual machines (VMs), network switches, containers, and others. API interactions that are smart and disclose resources are the responsibility of the provider. Terraform collaborates with a wide range of cloud providers.

5. What is the Terraform Work Process?

- Terraform init is used at the initial step to generate an operational directory including all Terraform configuration file contents.
- The Terraform plan, as the name implies, is to apply an execution strategy in a specific stage of development. It is significant since it will serve as the judging criteria to determine whether the expectations are reached.
- Terraform apply will guarantee that the plan is implemented within the timeframe specified in order to achieve the needed intended state of the infrastructure.
- Terraform destruction is the last stage in which this technology is utilized to remove all deployed resources.

6. Explain the workflow of the core terraform?

Core Terraform's workflow process consists of three steps:



- Write – Create infrastructure using coding
- Plan – Before executing the modifications, make a plan in advance to observe how they will seem.
- Apply – Apply to build an architecture that is repeatable.

7. Define Terragrunt?

- Terragrunt is a thin, covering layer that is used to cover terraform. This layer assists in the implementation of terraform-advocated and validated techniques. Terragrunt is useful for writing code on Terraform, but it is only available once. This reduces the need to develop code for each environment structure and deletes redundant code.
- It has several capabilities, such as lifespan, and it also gives flexibility when utilizing Terraform by supporting a continuous deployment process.

8. Explain the Terraform request flow architecture

<u>Command Line Interface (CLI):</u>

- Despite some preliminary bootstrapping in the root package, execution of the Terraform program immediately moves to one of the commanding package's "command" versions when a user launches it.
- The command names and command package types are mapped together and saved in the command. The file system of the repository contains the go file.
- The function of the command execution for these instructions is to read and examine any command-line parameters, command-line variables, and environment variables required for the provided operation and use them to construct a backend. aim of the operation The action is then transmitted to the backend that is currently selected.

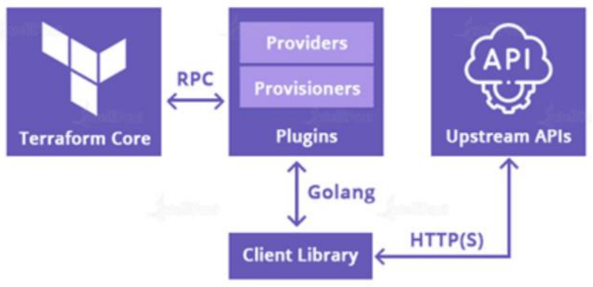
<u>Backend:</u>

A Terraform backend is in charge of several things:

- Deploy appropriate operations (e.g. plan, apply)
- Variables that have been defined in the workspace can be saved.
- To keep track of the current status

The local backend loads and performs initial processing/validation on the configuration specified in the operation using the config loader after first using a state supervisor (either statemgr. Filesystem if indeed the local backend is being used effectively, or an execution supplied via whatever backend is now being encased) to recover the present state for the working space stipulated in the procedure.

With these inputs and the additional parameters given via the procedure, it then creates a terraform—context object. The main item that performs actions is terraforming.



Architecture of Terraform

Configuration Loader:

Model types stand in for the upper configuration structure in package configurations. Config is used to symbolize a configuration (the root module and all of its child modules). The config load is the main access point, despite the fact that the configs package offers some limited configuration object generation capabilities.

The configload subpackage contains a loader. Intricacies of installing child modules (during Terraform init) and locating such modules when a configuration is loaded by a backend are handled by a loader. To create a single configuration, it loads all of the child modules recursively after taking the path to the root module.

State Manager:

- Frames of a workspace's Terraform state must be saved and retrieved by the state manager.
- The vast majority of managers handle the entire set of statemgr, whereas each manager handles a portion of the document's protocols.
- Complete the entire procedure
- There is no reason to construct a state manager that doesn't integrate all of statemgr; alternative function declarations often use the smaller protocols to define what operations the module may carry out on the state manager.

Graph Builder:

- The Context method calls a graph builder.
- A graph builder is utilized to illustrate the key phases in the process as well as their interdependencies.
- Due to variations in the graph-building procedure, each action has its own graph builder.
- A graph must be generated directly from the configuration for a "plan" operation, whereas a graph is constructed from the set of adjustments indicated in the plan being applied for a "apply" action.

Graph Walk:

- The graph walking method explores each vertices of the graph while taking into consideration what "happens after" the edges of the graph.
- Every vertices in the graph is evaluated such that "happens after" edges are taken into consideration.
- The graph walk method attempts to evaluate several vertices simultaneously.

Vertex Evaluation:

- Execution describes the process that takes place for each vertex throughout a graph walk.
- Execution carries out a set of arbitrary operations appropriate for the relevant edge type.
- Just before graph walk evaluates additional edges with "happens after" edges, one vertex must be completed successfully.
- The graph walk is stopped and the user is given the errors when one or more mistakes are made during evaluation.

9. What Terraform commands are the most useful?

Here are some useful Terraform Commands

- fmt
- init
- validate
- plan
- apply
- destroy
- output
- show
- state
- version

10. Explain Resource Graph in Terraform.

- A resource graph is a graphical presentation of the resources that are accessible. It allows for the simultaneous alternation and generation of separate resources.
- Terraform generates plans and refreshes the state by creating a plan for the graph's configuration. It promptly and successfully builds structure to assist us in understanding the drawbacks.

11. What do you understand by the term "Terraform Backend"?

Terraform provides certain configuration options to work at the backend level called "Terraform Backend," wherein developers are given a couple of options for the remote or local location to store and manage the built infrastructure. The interface to read and write the state data, along with storing the same file, is done with the help of the backend.

12. Explain the different types of Terraform Backends.

Terraform Backends extends as

Local Backend: This is the default backend that comes into the picture when no Terraform configurations are mentioned at the backend. This is used to store the state file on the machine itself, where the Terraform Backend is running.

Remote Backend: When state files are stored in a remote location, it is termed a remote backend. For example, a cloud object storage service is available on the cloud

13. Does Terraform provide options to deploy several providers?

Yes, Terraform provides options for multi-provider installations, which include SDN management, and also on-premise solutions like OpenStack and VMware.

14. Do we have any commands that completely remove the Terraform-managed infrastructure?

Yes, we can use the following command to completely remove the Terraform-managed infrastructure:

1 | [options] [dir] terraform destroy

15. What do you mean by "tainted resource"?

Terraform considers a resource to be tainted when the resource becomes inconsistent or enters a corrupted state. The terraform confirms that the resource is either damaged or has been downgraded due to any failure it has come across previously.

16. How can we discover plugins with the help of Terraform?

The interpretation of configuration files in the operational directory is done by Terraform using the command “Terraform init”. Post interpretation, the necessary plugins are identified, and then the search begins for the installed plugins in different locations. Additional plugins may also be downloaded when the requirement arises.

17. Explain the working of the following commands: Terraform -version Terraform fmt Terraform providers

Terraform -version – This command is used to identify which version of Terraform is installed.

Terraform fmt: This command can be used to rewrite the configuration files in a different format, like conical styles, etc.

Terraform providers: This command is used to seek information on the providers that are entitled to work on the current configuration.

18. Explain the command Terraform Validate.

This command is used to check the configuration files in the directory that are primarily focused on the configuration and, in turn, ignore any external services used, such as API providers. It validates the configuration to check whether the syntax is correct and consistent enough. Therefore, “Transform Validate” is the best way to validate the modules that are generally reusable.

19. How do you recover from a failed application in Terraform?

Before making any changes, it’s advisable to first save and commit your configuration in version control. This ensures that you have a backup in case you need to revert to the previous configuration. Furthermore, it is crucial to consistently resubmit previous versions of your code as new versions in your version control system.

20. Mention some of the use cases of Terragrunt.

Terragrunt works like an extension to Terraform. It can enhance the features offered by Terraform, along with making it more user-friendly. Following are the use cases where Terragrunt can be useful:

- **DRY (Don’t Repeat Yourself) Infrastructure Code**: Terragrunt helps reduce redundant code, hence making the developer’s life easy.
- **Remote State Management**: It can help simplify remote state management by helping you store it in different storage locations.
- **Environment-specific Configuration**: It can help segregate working environments like Dev, Staging, Prod etc.
- Dependencies Management across Terraform Configurations
- **Encrypted Variable Values**: It can make Terraform configurations more secure by encrypting variable values.
- **Automated Infrastructure Deployment**: Provides in-built support for CI/CD pipelines
- **Terraform Wrapper:** Terragrunt acts as a wrapper around Terraform commands, simplifying and enhancing the Terraform workflow. It adds features like automatic initialization and remote state configuration.

21. Explain the 'terraform graph' command.

It helps you create visual representations of the resource dependencies in Terraform. This can help you track which resources in the Terraform configuration file require a certain dependency.

22. What do you mean by a Terraform Directory?

Terraform directory houses all the configuration files in Terraform, such as main.tf, variables.tf, output.tf, etc. To initialize a Terraform directory, one has to type in the command 'terraform init' in the directory.

23. What do you mean by provisioners in Terraform?

The scripts that are part of resource creation and can be created either on a local machine or on a remote machine are executed using provisioners in Terraform. It can also be used to bootstrap a resource, clean up, and run/manage the configuration.

24. What do you understand by the term "Terraform Core"?

Terraform Core is considered the essential part of Terraform and is responsible for the fundamental functionalities of Terraform, which also include parsing the configuration files, creating an execution plan, and provisioning the infrastructure.

25. What do you mean by external data source in Terraform?

Terraform provides a special feature called an external data source feature, wherein users are allowed to use an external program to serve as a data source within the Terraform configuration. Additionally, an extra facility is provided where any kind of data can be shared across the platform using Terraform.

INTERMEDIATE QNS:

26. Define dependencies in Terraform?

You can use depends_on to identify the dependency. You may also use the relies on the parameter to indicate several resources, and Terraform will build the target resource when all of them have been built.

27. What do you mean by State File Locking?

- State file locking is a Terraform technique that prohibits multiple users from doing actions on the same state file at the same time. Once one user's lock on a state file is released, any other user who has a lock on it can act on it.
- This helps to prevent state file corruption. A backend operation is gaining a lock on a state file in the backend. If getting a lock on the state file takes longer than intended, a status message will be produced.

28. Mention some of the version control tools that Terraform supports.

Terraform supports the following version control tools:

- GitHub
- GitLab CE

- Bucket Cloud
- GitLab EE

29. Define Terraform cloud?

Terraform Cloud is software that enables teams to work together on Terraform. It provides features such as easy access to shared state and secret data, access controls for approving infrastructure changes, a private registry for sharing Terraform modules, detailed policy controls for governing the contents of Terraform configurations, and more to ensure that Terraform runs in a consistent and reliable environment.

30. What do you mean by Modules in Terraform?

In Terraform, a module is a container for various resources that are utilized in collaboration. The root module is required for any Terraform that includes resources listed in.tf files.

31. How to Ignore the Error Duplicate Resource when applying Terraform?

- Depending on the criteria, solutions might change.
- To terminate handling the resources, remove them from the Terraform code.
- By using terraform, you may destroy and regenerate resources through the API.
- Execute an importing action to delete the resources and the code that is attempting to rebuild them.

32. What are some of the notable applications that make Terraform useful?

Due to the general ability to terraform, the applications are highly remarkable and diverse in general.

The applications are as follows:

- Clusters of self-service
- Multi-tier application development
- Environment creation
- Resource allocation
- Creating a software demonstration
- Heroku app installation

33. What is the purpose of Terraform in DevOps?

- Terraform is a flexible tool for designing infrastructure using a proper code style. It is advantageous to have total orchestration control, similar to puppet and ansible.
- Terraform is an efficient and well-structured cloud platform that supports all of the main cloud providers such as GCP, Azure, and AWS.
- It is simple to manage due to its dynamic framework, which allows for easy configuration changes. It may also be easily switched from one supplier to another.
- It may be run on the masterless and client-only architecture mainframes with correct installation and use of all APIs.

34. What are the main characteristics of Terraform?

- Infrastructure as Code: Terraform’s high-level configuration language is used to describe your structure in logical file types that are human-readable.

- You can now create a blueprint that is editable, shareable, and reusable.
- Before making any infrastructure changes, Terraform develops an execution plan that outlines what it will perform and requests your consent. Before Terraform produces, upgrades, or destroys infrastructure, you may evaluate the changes.
- Terraform generates a resource graph while developing or modifying non-dependent resources. Terraform can now construct resources rapidly while also providing you with additional information about your infrastructure.

35. What do you mean by IAC?

IaC is an abbreviation for "Infrastructure as Code" IaC refers to a technique in which developers may run and provide computer data centres automatically rather than engaging in a physical process. Terraform IAC is an example of an IAC tool.

36. How do you define a null resource in Terraform?

The null resource follows the typical resource lifetime but does nothing else. The trigger parameter enables the setting of a subjective set of values that, if misrepresented, will result in the replacement of the reserve.

The principal use of the null resource is as a do-nothing container for arbitrary operations done by a provisioner.

37. Is Terraform suitable for on-premise infrastructure?

Yes, Terraform can be used to construct infrastructure on-premises. There are several services to choose from. You can choose whichever one best meets your requirements. Many individuals construct their own client Terraform providers; all that is necessary is an API.

38. What are some of the built-in provisioners available in Terraform?

Here is a list of Terraform's built-in provisioners:

- Salt-masterless Provisioner
- Puppet Provisioner
- File Provisioner
- Chef Provisioner
- Remote-exec Provisioner
- Local-exec Provisioner
- Habitat Provisioner

39. What are the Elements of Terraform architecture?

The Terraform architecture has the following characteristics:

- Expression Evaluation
- CLI (Command Line interface)
- Vertex Evaluation
- Sub-graphs
- State Manager
- Configuration Loader
- Graph Walk

- Graph Builder
- Backend

40. What are some of the most recent Terraform Azure Provider factors?

The most recent versions include additional data resources and Azurem batch certificate, which aids in certificate management. In networking, this resource is used to regulate the prefix. Bugs have been fixed, and azurerm app service has been improved.

41. What is the relevance of Terraform variables?

Variables in Terraform find their way into enhancing the reusability and flexibility of the configurations. Terraform extends the facility of acting as a parameter that allows the users to customize the configurations and set different values for various environments, allowing them to input the variables to add the adaptability layer to the infrastructural code

42. Differentiate between 'Terraform Plan' and 'Terraform apply'.

Terraform Plan: This command is used to establish an execution plan for the terraform configurations. It also examines the current state and the desired state to figure out the possibility of any potential problems before making any such changes.

Terraform Apply: This command is used to apply the changes that were proposed in the terraform plan.

43. Does Terraform allow you to manage resources across multiple cloud providers?

Yes, Terraform extends capabilities to manage resources available on multiple cloud providers, which include various cloud platforms like Google Cloud Platform, Microsoft Azure, etc.

44. How can you destroy the infrastructure created with Terraform?

Terraform allows you to destroy the infrastructure created with the help of Terraform using the 'Terraform Destroy' command. This command first reads the Terraform configuration, then creates a plan for destruction and throws a prompt to the user for final approval, after which the plan gets executed and changes are applied to the infrastructure.

45. How does Terraform handle State Management?

State files play a crucial role in monitoring the present condition of the infrastructure. Every detail regarding resources under the Terraform Management, which also includes their characteristics and interdependencies, is handled by the State Management. It acts as a Terraform's reference to adjust to the desired changes and achieve the desired state, while also maintaining the historical records of the infrastructure changes.

46. Does Terraform allow you to roll back the changes you make?

No, Terraform does not allow you to roll back the changes once they are made. However, the previous state of the infrastructure can be attained by reverting it to a previous state file. It is always important to maintain backups or different versions of the state file so rollbacks may be facilitated if needed.

47. What do you mean by the "plan refresh" process of Terraform?

This process involves comparing the configured resources and reading the current state of the infrastructure after analyzing the existing state and figuring out the adjustments required to align it with the desired state. In a nutshell, Terraform deeply analyzes what changes need to be made for better infrastructure.

48. What do you mean by terraform workspaces?

Terraform workspaces extend the facility to manage multiple instances of the single infrastructure existing in different environments. Workspaces enable us to maintain multiple instances while saving their respective states and variables without disturbing the Terraform configuration.

49. When are the Terraform workspaces brought into use?

Workspaces turn out to be useful when we need to maintain different sets of resources, including development, staging, and production environments, without disrupting the Terraform state configuration.

50. Differentiate between a Terraform provisioner and a resource.

In Terraform, a resource acts as a blueprint for creating a component of your infrastructure. Think of it like a machine or server on platforms such as AWS or Azure. These resources define the desired structure of your infrastructure. Now, when it comes to ensuring everything is perfectly set up, that's where provisioners come into play. They are like the hands-on workers who execute scripts or commands on your resource after it has been created or updated. This is useful for tasks such as software installation, configuration management, and application deployment. So essentially, resources shape your infrastructure, while provisioners fine-tune it to meet your requirements.

ADVANCED QNS:

51. Are callbacks possible with Terraform on Azure

Yes, callbacks are possible with Terraform on Azure using Azure Event Hubs. Terraform's AzureRM provider provides the necessary functionality to integrate with Azure Event Hubs and trigger callbacks based on specific events.

52. What is Terraform D?

Terraform D is a declarative syntax for writing Terraform configurations. It's an alternative to the traditional HCL syntax and aims to provide a more concise and readable way to define infrastructure resources. Terraform D is still under development, but it has the potential to simplify Terraform configurations and make them easier to understand and maintain.

53. Why is Terraform used for DevOps?

Terraform is widely used in DevOps because it enables infrastructure as code (IaC), which means that infrastructure is defined and managed using code rather than manual configuration. This approach has several advantages for DevOps, including:

Automation: Terraform configurations can be automatically applied and set up, which can save time and reduce errors.

Reproducibility: Infrastructure can be easily replicated and recreated from code, which ensures consistency across different environments.

Version control: Terraform configurations can be stored in version control systems, which allows for easy tracking of changes and rollbacks

54. Explain the uses of Terraform CLI and list some basic CLI commands.

Terraform CLI (command-line interface) is the primary tool for interacting with Terraform. It provides a number of commands for managing infrastructure, including:

- **terraform init:** Initializes a Terraform directory and downloads the necessary providers.
- **terraform plan**: Generates an execution plan that shows the changes that Terraform will make to the infrastructure.
- **Terraform apply**: Applies the changes in the execution plan to the infrastructure.
- **Terraform destroy**: Destroys the infrastructure managed by Terraform.

55. Name all version controls supported by Terraform.

Terraform supports multiple version control systems, such as Git, Mercurial, Subversion, and Perforce. Git is commonly used due to its flexibility, branching, and collaboration features, making it the preferred choice for managing Terraform configurations across teams.

56. Give the terraform configuration for creating a single EC2 instance on AWS.

```
provider "aws" {
  region = "us-west-2"   # Replace with your desired region
}
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

This snippet uses the AWS provider to specify the region and creates an EC2 instance with a specified AMI and instance type. Adjust the parameters to fit your requirements.

57. How does Terraform handle the drift detection of infrastructure state, and what actions can it take?

Terraform detects infrastructure drift by comparing the current state with the expected state described in configuration files. It offers commands like “terraform plan” to identify changes and “terraform apply” to reconcile and bring the actual state in line with the desired state.

58. What is a Private Module Registry?

A private module registry is a repository where organizations can store and manage their own Terraform modules. This allows them to control access to their modules and ensure that only authorized users can use them. Private module registries can be hosted on-premises or in the cloud. Some popular private module registries include GitHub Packages, Terraform Cloud, and HashiCorp Vault.

59. Can you provide a few examples that we can use for Sentinel policies?

Sentinel policies are used to define who can use Azure resources and how they can be used. They can be used to enforce security, compliance, and business policies. Here are a few examples of how Sentinel policies can be used:

- **Enforce the least privilege**: Ensure that users only have access to the resources they need to do their jobs.
- **Prevent unauthorized access**: Block access to resources from unauthorized users.
- **Enforce compliance**: Ensure that resource usage complies with company policies and regulations.

60. Which value of the TF_LOG variable provides the most verbose logging?

The most verbose logging level in Terraform is TRACE. This level will log all Terraform messages, including debug messages and provider plugin messages. To set the logging level to TRACE, use the following command:

TF_LOG=TRACE terraform plan

The other logging levels, in order of increasing verbosity, are **DEBUG, INFO, WARN, and ERROR.**

61. What are the main key responsibilities of Terraform Core?

There are certain key responsibilities of the Terraform Core:

- Interpolation of module and configuration file
- Constructing resource graphs
- Establish communication between plugins and RPC
- Managing the state resources

62. Why do DevOps teams prefer Terraform?

Terraform operates as a user-friendly configuration language similar to JSON. The syntaxes provided are simple, which in turn makes them user-friendly. This language empowers the DevOps teams to create configurations for the infrastructure effortlessly. Furthermore, these configurations can be implemented across different clouds and data centers, which provides a versatile solution to build and manage infrastructure setups.

63. What do you mean by a Terraform provider, and how can you create a custom provider?

The plugins that interact with infrastructure APIs and enable the facility to manage the resources are called Terraform providers. The available resources, their properties, and the actions to be performed are defined. To create a custom provider, the developer needs to develop a plugin that abides by the TerraformProvider Protocols, implying the necessary operations for resource management.

64. Explain 'Terraform as a service'.

It refers to the practice of providing Terraform functionality as a managed service. There are certain benefits to using Terraform as a service, including:

- Infrastructure provisioning is simplified and managed.
- DevOps tools are seamlessly integrated with the workflows.

- It provides centralized control and visibility across multiple environments.

65. How can you structure the modules more effectively?

The following best practices are advised to be followed to structure the modules:

- Keep the modules focused and single-purpose.
- Clear documentation must be provided.
- Flexibility and reusability should be maintained.
- Input variables are defined to customize the module behavior.

66. How is infrastructure testing implemented and validated?

Different tools and practices can be brought to use for infrastructure testing and validation, such as:

- The Terraform validate command can be used to check for configuration errors and syntaxes.
- Automated testing frameworks can be used by employees like Terratest or Kitchen-Terraform to execute infrastructure tests.
- Lining tools like TFLint can be incorporated.
- End-to-end testing and leveraging integration can be done to validate the behavior of the infrastructure.

67. Explain immutable infrastructure.

When infrastructures are termed disposable and no modification is allowed once they are provisioned, those infrastructures are called immutable infrastructures. Immutable infrastructures are supported by Terraform by promoting the recreation of resources. Commands such as 'destroy' and 'apply' can be used to destroy and recreate every change, ensuring a consistent and predictable infrastructure.

68. How do you handle complex dependency management in Terraform?

The following steps can be followed to handle complex dependency management:

- Leverage implicit and explicit dependencies that are predefined in configurations.
- Use 'terraform state' command that helps us modify the resource ordering.
- Break down complex configurations into smaller modules so that the dependency is simplified.

69. What do you mean by Terraform State?

The state that represents the current state within the managed infrastructure is called a Terraform State. The information related to its resources, properties, and dependencies is also managed under the terraform state.

70. How do you manage the state in a team environment?

The following steps can be followed to manage the state in a team environment:-

- A shared remote backend can be used to maintain and store the state file, which also enables the option of collaboration.
- A state-locking mechanism can be maintained to prevent irrelevant modifications
- Clear ownership and access controls can be defined.
- Implement a state versioning strategy for better traceability.