Terraform state persistency schema

A structure that defines how Terraform tracks the state of infrastructure resources.
Terraform stores the current state of infrastructure in a state file (`terraform.tfstate`). This file records metadata, resource dependencies, and attributes. When changes are applied, Terraform compares the current configuration with the stored state to generate an execution plan. This file can be local or remote (e.g., in an S3 bucket), enabling collaboration and consistency.

AWS S3 storage service

Amazon Simple Storage Service (S3) is an object storage service.
S3 stores data as objects in buckets. Each object has a key (identifier), data, and metadata. It provides high durability and scalability and supports operations like PUT, GET, and DELETE via API or SDK. Common use cases include static website hosting, backup, and data lakes.

- **PaaS - Platform As a Service:** A cloud computing service that provides a platform allowing customers to develop, run, and manage applications without the complexity of infrastructure management.
- **SaaS** - Software As a Service: A software distribution model where applications are hosted by a service provider and made available to customers over the internet.

- **IaaS** - Infrastructure As a Service: A cloud computing service model that provides virtualized computing resources over the internet, allowing users to rent IT infrastructure.

- **FaaS** - Function As a Service: A serverless computing execution model where functions are executed in response to events without the need for maintaining server infrastructure.

- **IaC** - Infrastructure as Code: The practice of managing and provisioning computing infrastructure through machine-readable script files, rather than physical hardware configuration.

DMN - Decision Model and Notation

A standard notation for decision modeling in the business process management field, enabling visual representation of decision logic.

BPM - Business Process Management

A **discipline** involving methods, techniques, and tools to design, enact, monitor, and optimize business processes.

BPMN - Business Process Model and notation

A **graphical representation** for specifying business processes in a business process model, widely used for workflow automation.

Processes:
- A **BPMN descriptive process** is a _visual model_ used mainly for documentation or communication, not meant to be run by a process engine.
- A **BPMN executable process** is a _machine-readable_ version with precise technical details (e.g., task types, service calls) that can be executed by engines like Camunda.

Elements:

- **BPMN Activity**: An element within a BPMN diagram representing work that needs to be done.
- **BPMN Gateway**: A decision point or forking mechanism in a BPMN diagram, controlling the flow of sequences.
- **BPMN Event**: An occurrence that triggers a sequence or alters the flow within a BPMN process.
- **BPMN Pool**: A container within a BPMN diagram representing a participant in a business process (e.g., an organization or system), defining boundaries of process responsibility.
- **BPMN Lane**: A sub-partition within a BPMN Pool that typically representing roles or departments. Lanes help organize tasks assigned to different actors within the same participant.

DMN versus using BPMN gateways

	DMN	BPMN Gateways
Advantages	**Business-friendly**: Easier for business analysts to understand and maintain decision logic.	**Process Control**: Gateways are better for controlling the flow based on conditions and events.
	Separation of Concerns: Keeps decision logic separate from process flow, promoting modularity.	**Visualization**: Makes branching logic visible in the process model.
	Reusability: DMN tables can be reused across multiple processes or scenarios.	-
Pitfalls	**Limited Flow Control**: Cannot handle complex process routing; must be combined with BPMN.	**Harder to Maintain Logic**: Complex conditions can become hard to manage and understand.
	Learning Curve: Requires understanding of decision modeling and table syntax.	**Duplication**: Decision logic may be duplicated across gateways instead of centralized.
	Tooling Limitations: Some platforms may have limited DMN tooling or debugging support.	-

Workflow systems

Systems that manage and automate structured business processes, ensuring tasks follow predefined sequences towards a specific goal.

Business process

A set of structured activities or tasks that produce a specific service or product.
Defined using models (e.g., BPMN), executed in sequence or parallel, with decisions, loops, and event-based transitions. It can include automated services and human involvement and aims at achieving business goals efficiently.

Business process engine

A software system that executes and manages business process models.
It interprets models (e.g., BPMN), executes tasks (manual or automated), tracks state, and supports events, timers, gateways, and human interactions. It ensures workflows follow defined logic and transitions across states until completion.

Camunda form

A UI form associated with user tasks in Camunda processes.Defined using Camunda Forms or external form frameworks (e.g., Angular, React), the form collects input from users and submits it back to the process engine to move the workflow forward.

Camunda http-connector

A built-in service task type in Camunda to make HTTP requests.
Configured using process model properties (e.g., URL, method, headers, payload), the connector sends an HTTP request during process execution and uses the response to influence the process flow or variables.

Camunda user interaction task

A BPMN task where a human user is required to perform an action.
When a process reaches this task, it pauses and assigns the task to a user (based on roles or groups). The user interacts via the task list UI or an external app. Once completed, the process resumes.

Camunda service task

An automated task in BPMN executed by the engine without user involvement.
Executes external logic such as calling REST APIs, invoking Java delegates, or using connectors.
Used for automation, like sending emails, triggering jobs, or processing data.

Pull and push for Camunda task execution

- **Pull**: External workers _poll_ the process engine (e.g., via Camunda External Task API) to fetch and execute tasks.
- **Push**: Tasks are _directly assigned_ to users or systems (e.g., user tasks in forms or service tasks calling Java delegates), without polling.

-	Pull Model	Push Model
Advantages	**Flexibility**: Workers can fetch and execute tasks at their own pace.	**Low Latency**: Tasks are immediately pushed to users or systems for execution.
	Scalability: Easy to horizontally scale by adding more workers.	**Simplicity**: Easier to understand and implement, especially for human tasks.
	Decoupling: External workers are loosely coupled to the engine.	-
Pitfalls	**Polling Overhead**: Continuous polling can increase load on the engine.	**Tight Coupling**: Requires synchronous communication or close integration.
	Latency: Delay between task availability and worker fetching it.	**Scalability Issues**: Less suitable for high-throughput automated task execution.
	Complexity: Requires additional infrastructure to manage workers and retries.	-

Camunda business key

An identifier used to uniquely identify and manage instances of processes in the Camunda BPM platform.

Correlation ID of a process using Camunda business key

A unique identifier used to correlate messages or interactions with specific process instances.
The `businessKey` is set at process start and can be used to find or correlate events (like messages) to that instance. Useful for tracking, debugging, or linking external events.

Authorization of a business process engine

The mechanism that controls user or system permissions within the engine.
It verifies whether a given user, role, or service account has rights to perform certain actions (e.g., start a process, complete a task). This can involve role-based access control (RBAC) and integration with identity providers for authentication.

Access Authorization

The process of determining user privileges for accessing a resource or service.
After authentication, the system checks policies or roles to decide if the user can access a resource. This often includes permission levels (read, write, execute) and is enforced via mechanisms like access tokens, ACLs, or OAuth scopes.

Identity management

The administration of user identities and their associated access permissions.
It involves user provisioning, authentication, authorization, and lifecycle management. Identity providers (e.g., Keycloak, AWS IAM) handle user credentials, groups, and roles, often integrated with SSO, MFA, and directory services (LDAP, Active Directory).

API Gateway - Application Programming Interface gateway

An architectural pattern that provides a single entry point for clients to access multiple APIs, ensuring security, throttling, and monitoring.

REST API - REpresentational State Transfer – Application Programming Interface

A software architectural style for designing networked applications, emphasizing stateless communication over HTTP.

REST API

A web-based API following the principles of Representational State Transfer.
REST APIs use standard HTTP methods (GET, POST, PUT, DELETE) to manipulate resources identified by URIs. They are stateless, meaning each request contains all information needed for processing, and responses are typically formatted in JSON or XML.

Microservice

A software architecture pattern where complex applications are composed of small, independent processes communicating through APIs.

Serverless computing

A cloud computing execution model where cloud providers dynamically manage the allocation and provisioning of servers.

Serverless functions

Stateless pieces of code that run in response to events without the need for server management. They are executed on-demand, scaling automatically based on the number of requests.

Advantages	Pitfalls
Cost Efficiency: Pay only for actual usage, not idle time.	**Cold Starts**: Initial latency when functions are invoked after inactivity.
Scalability: Automatically scales based on demand without manual intervention.	**Execution Limits**: Functions may have timeouts, memory, and compute limits.
Simplicity: Developers focus on writing code without worrying about infrastructure.	**Observability**: Debugging and tracing can be more difficult than in traditional apps.
-	**Vendor Lock-in**: Tied to the specific cloud provider's ecosystem and tooling.

Serverless API invocation process. What is cold start and warm execution?

The process of invoking APIs in serverless environments like AWS Lambda.
On first invocation or after a period of inactivity, the platform initializes the function's runtime environment (cold start), which causes latency. After initial warm-up, subsequent invocations reuse the runtime (warm execution), resulting in lower latency. Cold starts impact performance and are influenced by factors like language runtime, memory size, and concurrency.

Microservices framework and serverless functions

- A **microservices framework** structures applications as a _set of independent services_ that run continuously, often on containers or VMs, communicating via APIs.
- **Serverless functions** are _event-driven, stateless_ functions executed on-demand. They scale automatically and only run when triggered, reducing idle compute time.

Containers and serverless functions

- **Containers** package applications with all their dependencies and run _persistently_ or long-lived on orchestrators like Kubernetes.
- **Serverless functions** are _ephemeral_, invoked per event/request, and managed entirely by the cloud provider with no server visibility or lifecycle control.

Kubernetes

An open-source container orchestration platform for automating deployment, scaling, and management of containerized applications.
It abstracts infrastructure, schedules containers in Pods, handles service discovery, load balancing, storage, and rolling updates via declarative YAML configurations.

Kubernetes Pod

The smallest deployable unit in Kubernetes, representing a set of one or more containers sharing network and storage resources.

Docker

A platform for developing, shipping, and running applications using container virtualization technology.

Docker image

A lightweight, standalone, executable package that includes everything needed to run a piece of software, including code, runtime, libraries, and dependencies.
Is a _read-only template_ with the application and its dependencies. It's the blueprint for creating containers.

Docker container

An instance of a Docker image that can be run as an isolated process, providing a consistent environment for applications.
Is a _running instance_ of a Docker image. It’s the live, executable unit that includes the application process and its isolated environment.

Kubernetes versus Docker Compose

-	Kubernetes	Docker Compose
Advantages	**Production-Ready**: Designed for scalable, resilient, and orchestrated deployments.	**Simplicity**: Easy to use and quick to set up multi-container applications.
	Advanced Features: Includes load balancing, autoscaling, rolling updates, and health checks.	**Local Development**: Ideal for testing and development environments.
	Ecosystem Support: Integrates with cloud-native tools for monitoring, logging, and CI/CD.	
Pitfalls	**Complex Setup**: Steeper learning curve and more components to manage.	**Limited Orchestration**: Not suitable for managing large-scale production workloads.
	Resource Intensive: Higher system requirements compared to Docker Compose.	**Lack of Features**: Missing features like autoscaling, secrets management, or service discovery at scale.
	Overkill for Small Projects: Kubernetes is often too complex for simple or local development setups.	

Virtualization and containerization

- **Virtualization** runs _multiple operating systems_ on a single physical machine using hypervisors. Each virtual machine has its own OS and kernel.
- **Containerization** runs _multiple isolated applications_ on the same OS kernel, sharing it. Containers are lighter and faster to start than VMs.