Security Governance: This category includes policies designed to protect software systems by defining and enforcing security-related rules. These rules ensure secure configurations, access restrictions, and appropriate behavior of infrastructure and services.

• Access Control: This sub-category includes policies that restrict or grant access to specific resources based on roles, identity, or contextual factors. Examples include authorization policies for users, groups, or services that manage who can access what under what conditions.

• Configuration Validation: These policies ensure that individual resources conform to specific security, structural, or operational constraints before being accepted or deployed. It focuses on validating fields such as volume types, base images, identity settings, and configuration correctness to prevent misconfigurations and enforce secure defaults.

• Secrets Management: These policies focus on securing sensitive data such as API tokens, credentials, and keys. Policies in this sub-category enforce best practices for handling secrets—e.g., ensuring secrets are not hardcoded or are retrieved securely from a secrets manager.

• Network Management: These policies regulate traffic flow, IP whitelisting, and connectivity rules within a system. They often include firewall rules, port restrictions, and control over ingress/egress configurations to reduce attack surfaces.

• Resource Management: These policies enforce constraints on the creation, usage, and allocation of system resources such as CPU, memory, containers, VMs, disk, clusters and Kubernetes pods. Typical policies ensure the creation, usage or allocation of secure system resources.

• Security Review Compliance: Contains policies that verify whether certain conditions indicating security review processes have been met. This ensures traceability and accountability during deployment and operational workflows.

• Vulnerability Management: These policies aim to detect and prevent the deployment of software with known vulnerabilities or missing updates. This may include policies enforcing up-to-date package versions. or disallowing insecure dependencies.

• Actions Restrictions: This sub-category focuses on preventing the execution of specific actions or commands that are considered harmful, unauthorized, or non-compliant with security policies. For example, a policy might deny execution of any command listed in a predefined denylist within a cloud-native container environment.

• Workloads Management: This sub-category focuses on the configuration and enforcement of operational constraints for workloads. For example, policies may deny the deployment of workloads that automatically mount service account tokens, to enhancing runtime security.

Compliance Governance: This taxonomy category ensures systems adhere to organizational policies, industry standards, and legal requirements. It covers internal best practices, external regulations, and service-specific constraints.

• Resource Compliance: Policies that enforces structural or content rules on resource definitions to comply with organizational standards. These policies verify whether a resource instance meets defined compliance criteria—typically structural, security, or configuration requirements. These policies operate at the level of resource correctness, such as ensuring a Kubernetes Pod has a required label, a Terraform file is in valid format, or a YAML config is syntactically and semantically sound.

• Service Compliance: These policies ensure that services meet specific operational, security, or performance requirements—e.g., ensuring logging or monitoring is enabled for critical services.

• Third-party License Compliance: Policies that check whether third-party systems and dependencies comply with organizational licensing standards. These policies are crucial in contexts subject to legal compliance and audits.

• Standards Enforcement: This category enforces adherence of a particular technology to its best practice implementations, often to improve maintainability, reusability, and operational consistency. It includes policies that ensure consistent naming, approved module usage, tagging standards, and container optimization across systems. example, a common policy requires the inclusion of yum clean all after yum install commands in Dockerfiles to ensure optimized image sizes and prevent unnecessary cache buildup.

Cost Optimization: This category includes policies aimed at reducing unnecessary cloud spending. Examples include enforcing resource quotas, limiting compute capacity, and identifying idle or underutilized resources.

Workflow Automation: This category includes policies that trigger actions or workflows automatically, such as enforcing compliance during CI/CD, triggering audits, or provisioning of clouds infrastructure, resource allocation.

Deployment Governance: This category includes policies that govern the secure, compliant, and auditable deployment of software systems. It ensures that deployments are performed by authorized entities, under predefined conditions, and in alignment with operational and security requirements—such as validating artifact integrity, enforcing configuration constraints, and controlling deployment permissions.

• Key Management: This sub-category focuses on the creation, monitoring, and lifecycle control of deployment- related keys. For example, a policy might trigger alerts when a new deploy key is created, ensuring visibility and accountability over sensitive credential usage during deployment workflows.

• Access Control: This sub-category focuses on regulating who can perform which deployment actions and under what conditions. Policies in this group ensure that only specific users or agents with verified identities and at- tributes are permitted to initiate deployment operations.