US 20250298633A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2025/0298633 A1

McPeak et al. (43) Pub. Date: Sep. 25, 2025

(54) **RESOURCE PARAMETER RECOMMENDATION ENGINE BASED ON DEPENDENT RESOURCES AND GUARDRAIL CONFLICTS**

(71) Applicant: **Resourcely Inc.**, San Jose, CA (US)

(72) Inventors: **Travis MacLeod McPeak**, San Jose, CA (US); **Alaeddin Saleh Abdelrahman Almubayed**, Berkeley, CA (US); **David Robinson Bild**, Chicago, IL (US); **Nathaniel Glass**, Bellevue, WA (US); **Daniel James Rutledge**, Austin, TX (US)

(21) Appl. No.: **19/083,300**

(22) Filed: **Mar. 18, 2025**

Related U.S. Application Data

(60) Provisional application No. 63/567,848, filed on Mar. 20, 2024.

Publication Classification

(51) **Int. Cl.**

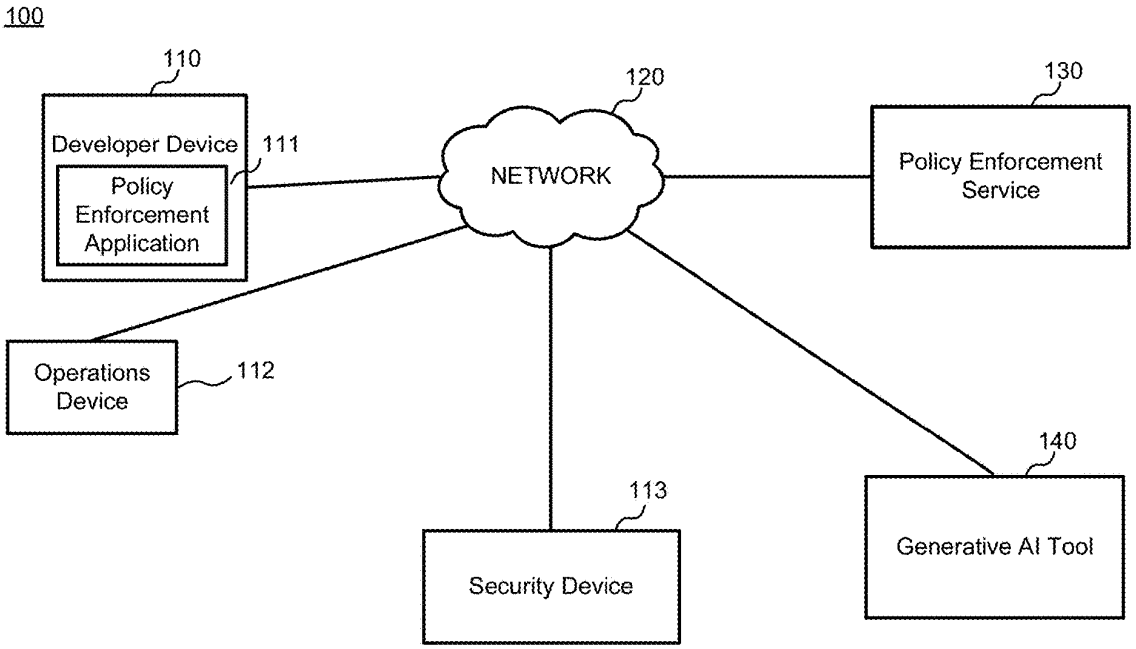
G06F 9/445	(2018.01)
G06F 3/04847	(2022.01)
G06F 9/451	(2018.01)

(52) **U.S. Cl.**

CPC ***G06F 9/44505*** (2013.01); ***G06F 3/04847*** (2013.01); ***G06F 9/451*** (2018.02)

(57) **ABSTRACT**

Systems and methods are disclosed herein for enforcing guardrails of a primary resource when configuring parameters having downstream resources. An application displays a user interface having configuration fields for configuring parameters of a resource, the resource having a blueprint defining guardrails. The application determines a set of configuration options for a parameter of the parameters of the resource, and determines, for each option of the set of configuration options, a dependent resource. For each dependent resource, the application retrieves a blueprint for the dependent resource and determining whether the blueprint for the dependent resource violates the guardrails. The application determines a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails, and displays a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset.



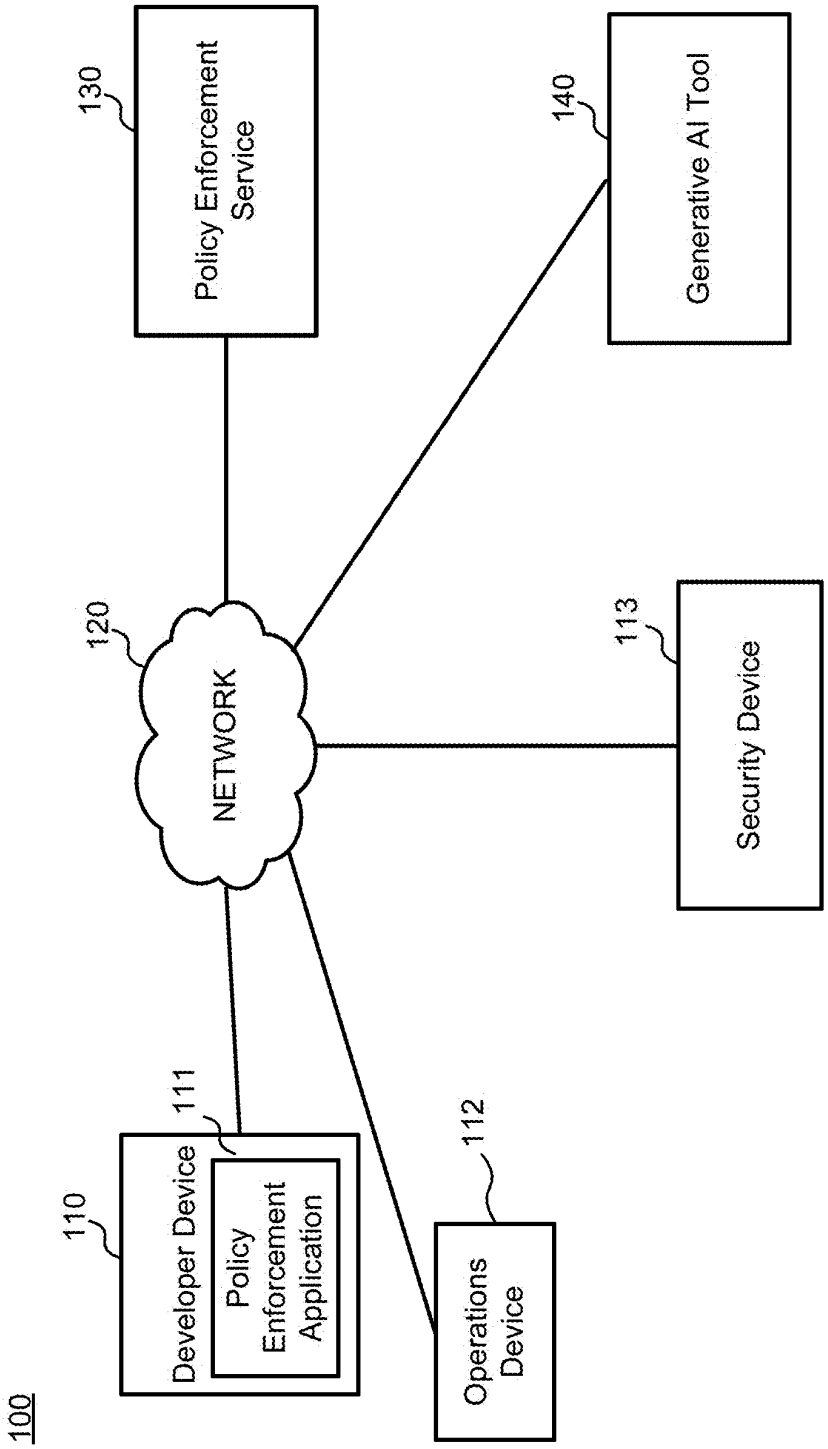

100
110
Developer Device
111
Policy Enforcement Application
112
Operations Device
120
NETWORK
130
Policy Enforcement Service
140
Generative AI Tool
113
Security Device

FIG. 1

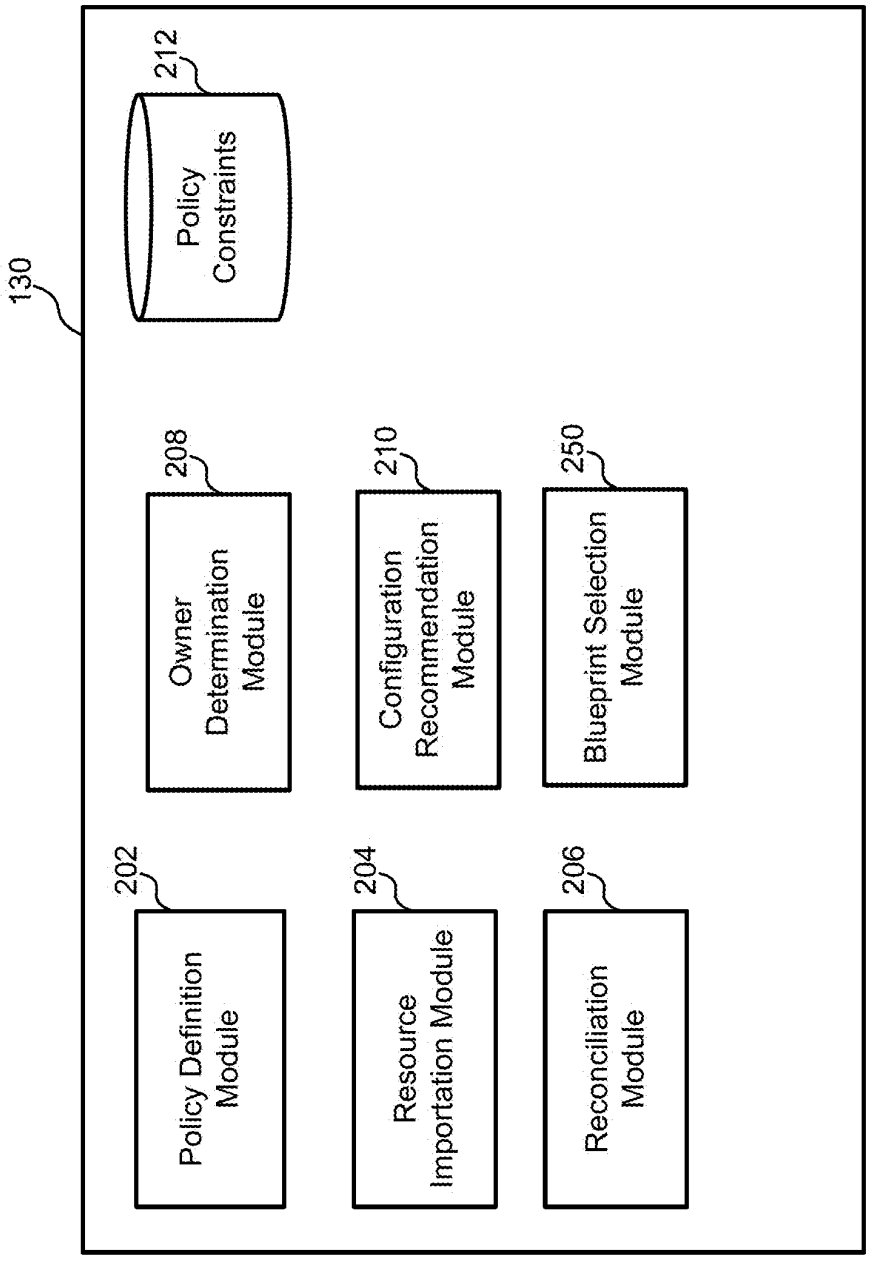

130
212
Policy Constraints
208
Owner Determination Module
210
Configuration Recommendation Module
250
Blueprint Selection Module
202
Policy Definition Module
204
Resource Importation Module
206
Reconciliation Module

FIG. 2

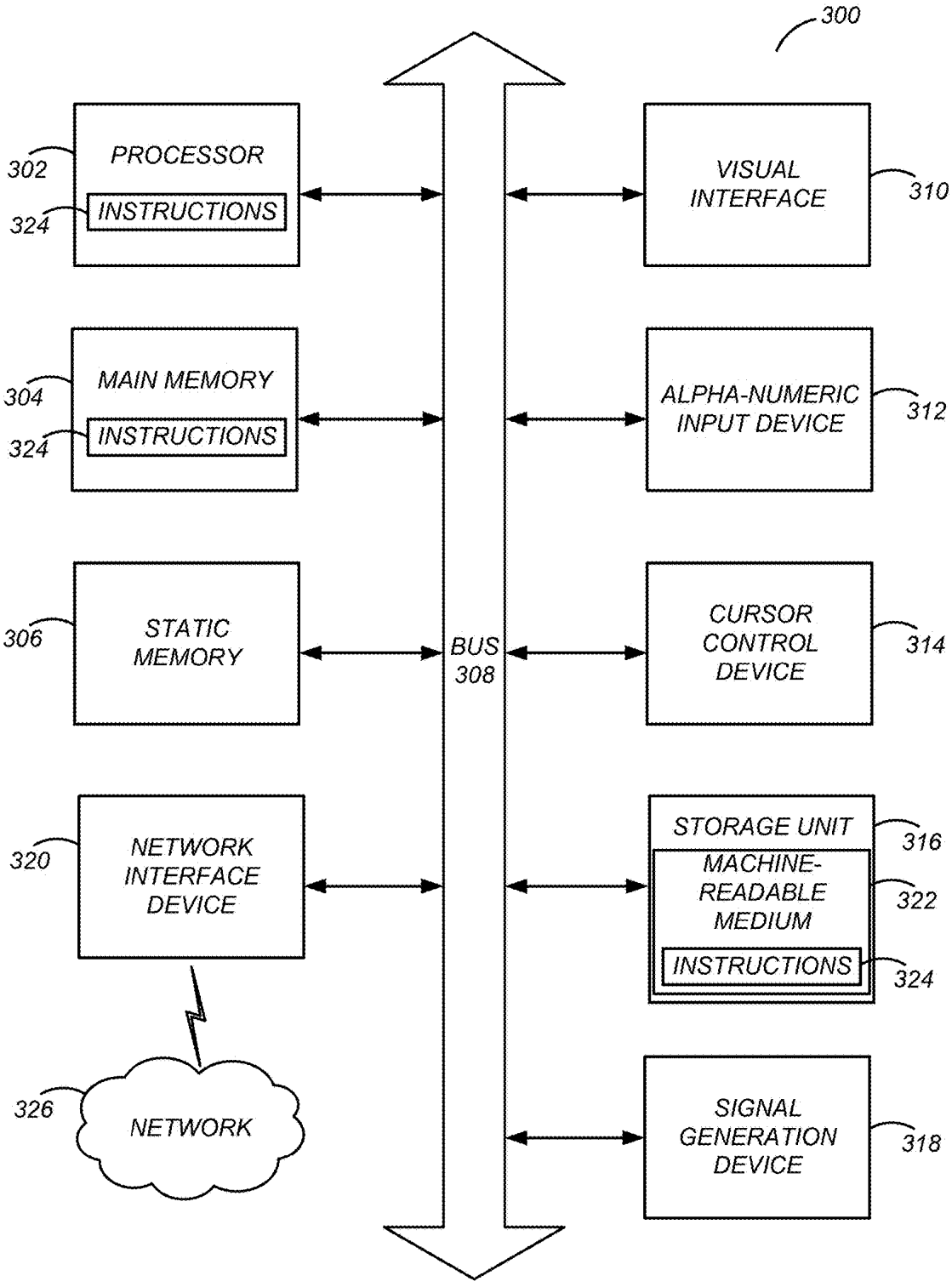

300
PROCESSOR
302
INSTRUCTIONS
324
MAIN MEMORY
304
INSTRUCTIONS
324
STATIC MEMORY
306
NETWORK INTERFACE DEVICE
320
NETWORK
326
BUS
308
VISUAL INTERFACE
310
ALPHA-NUMERIC INPUT DEVICE
312
CURSOR CONTROL DEVICE
314
STORAGE UNIT
316
MACHINE-READABLE MEDIUM
322
INSTRUCTIONS
324
SIGNAL GENERATION DEVICE
318

FIG. 3

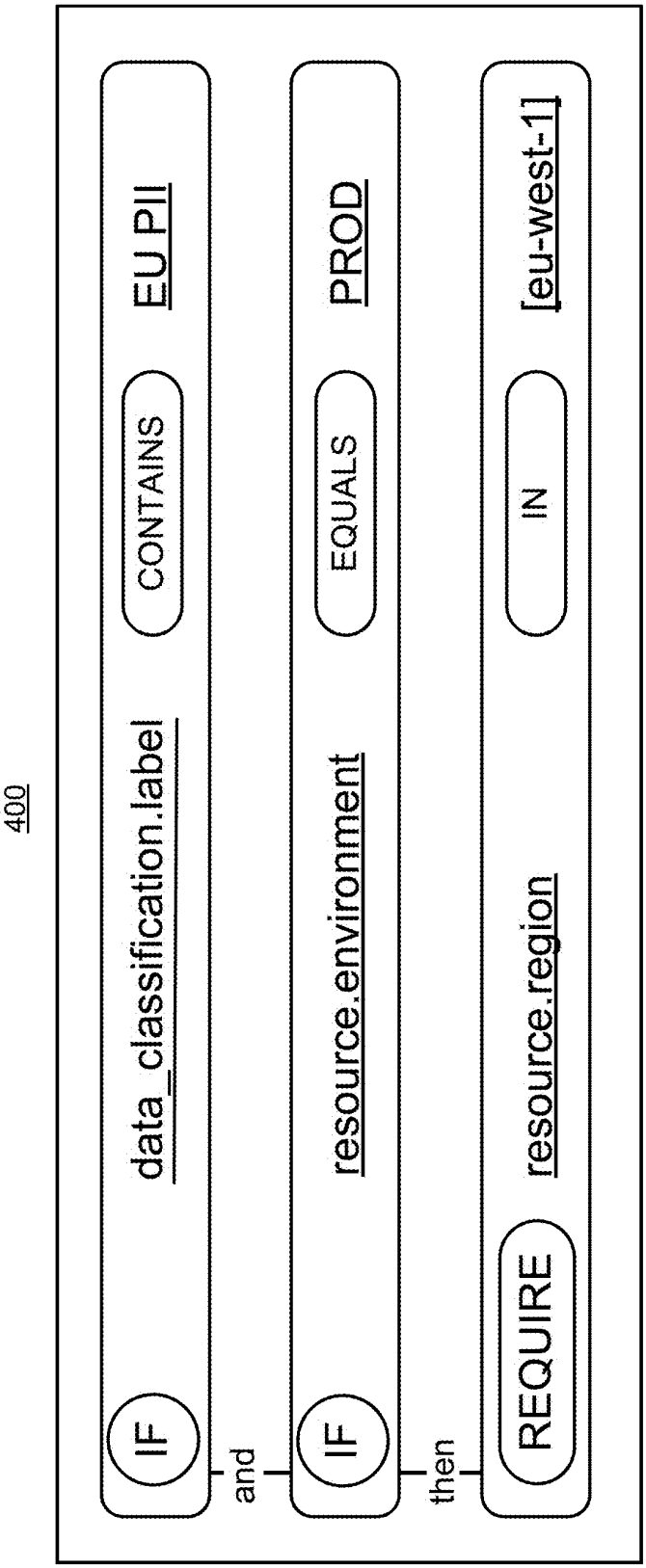

400
IF
data_classification.label
CONTAINS
EU PII
and
IF
resource.environment
EQUALS
PROD
then
REQUIRE
resource.region
IN
[eu-west-1]

FIG. 4

500

$ resourcely create s3 "my-bucket"

Hi, in order to create a secure S3 bucket, we need to collect more context!

What is your team name?

What is the purpose of this bucket?

Is your bucket going to store any of the following data? (Select all that Apply)

[X] PII

[X] EU citizen data

[ ] Financial Data

[ ] Corporate Data

>>Based on what you provided so far, this S3 bucket required to be deployed in EU region, other regions require approval

Which region would you like to deploy the bucket at?

[X] eu-west-1 (default – does not require approval)

[ ] Other (requires approval)

What bucket access would you like to create?

[X] Private (default – does not require approval)

[ ] Public (requires approval)

Thank you, please hold on!
generating terraform and submitting github pull request . . . !

PR link: https://github.com/Resourcely-Inc/terraform/pull/15

FIG. 5

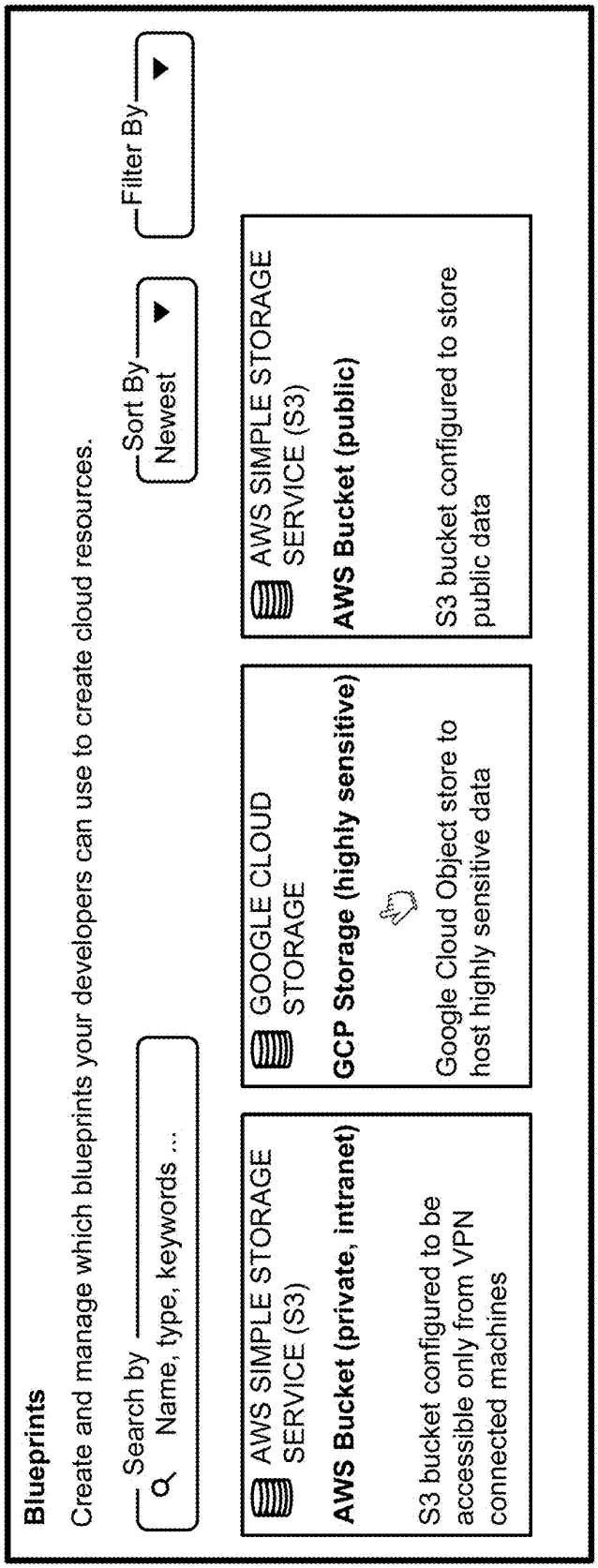

600
Blueprints
Create and manage which blueprints your developers can use to create cloud resources.
Search by
Name, type, keywords ...
Sort By
Newest
Filter By
AWS SIMPLE STORAGE SERVICE (S3)
AWS Bucket (private, intranet)
S3 bucket configured to be accessible only from VPN connected machines
GOOGLE CLOUD STORAGE
GCP Storage (highly sensitive)
Google Cloud Object store to host highly sensitive data
AWS SIMPLE STORAGE SERVICE (S3)
AWS Bucket (public)
S3 bucket configured to store public data

FIG. 6

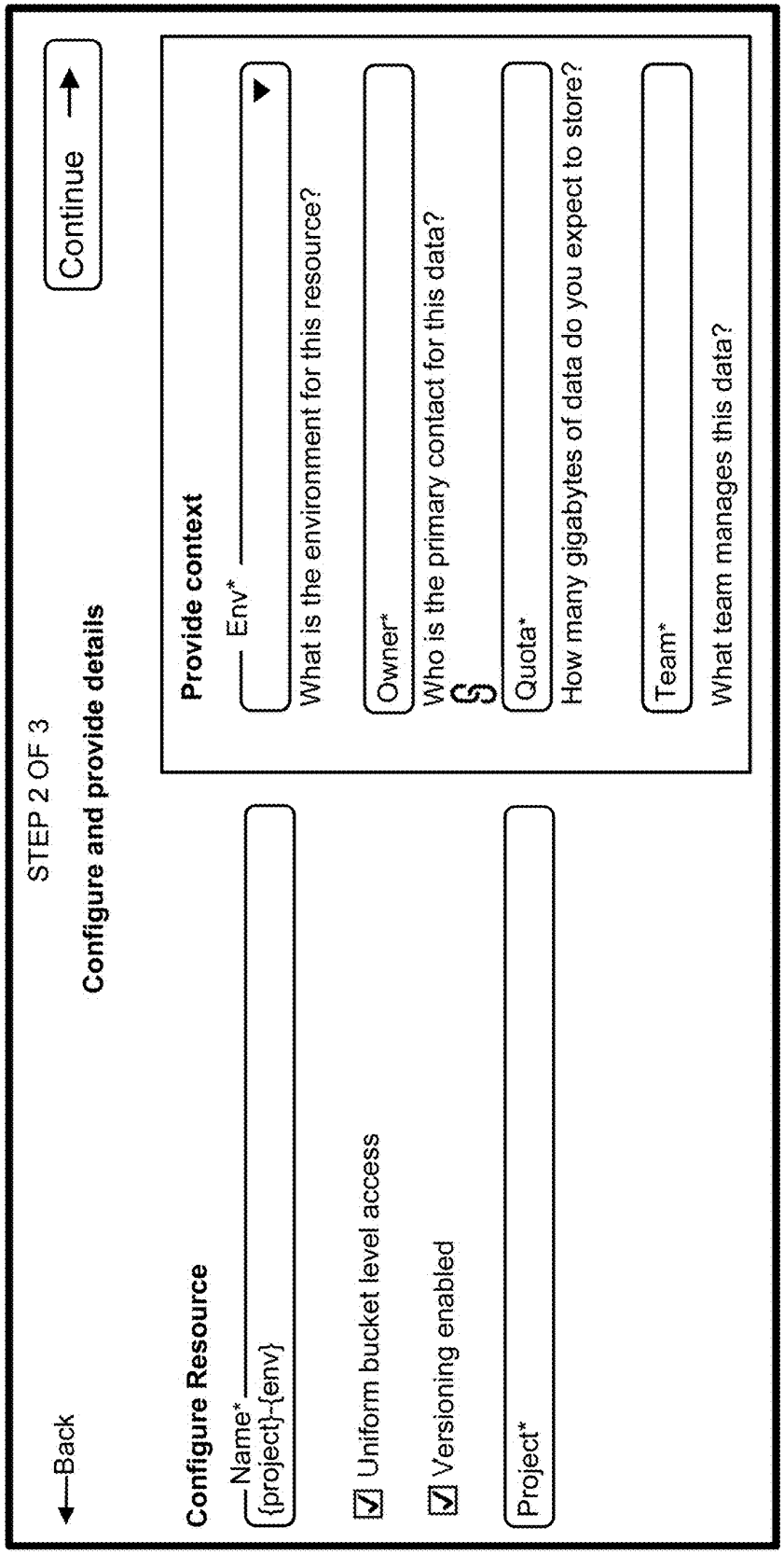

700
Back
STEP 2 OF 3
Configure and provide details
Continue
Configure Resource
Name*
{project}-{env}
Uniform bucket level access
Versioning enabled
Project*
Provide context
Env*
What is the environment for this resource?
Owner*
Who is the primary contact for this data?
Quota*
How many gigabytes of data do you expect to store?
Team*
What team manages this data?

FIG. 7

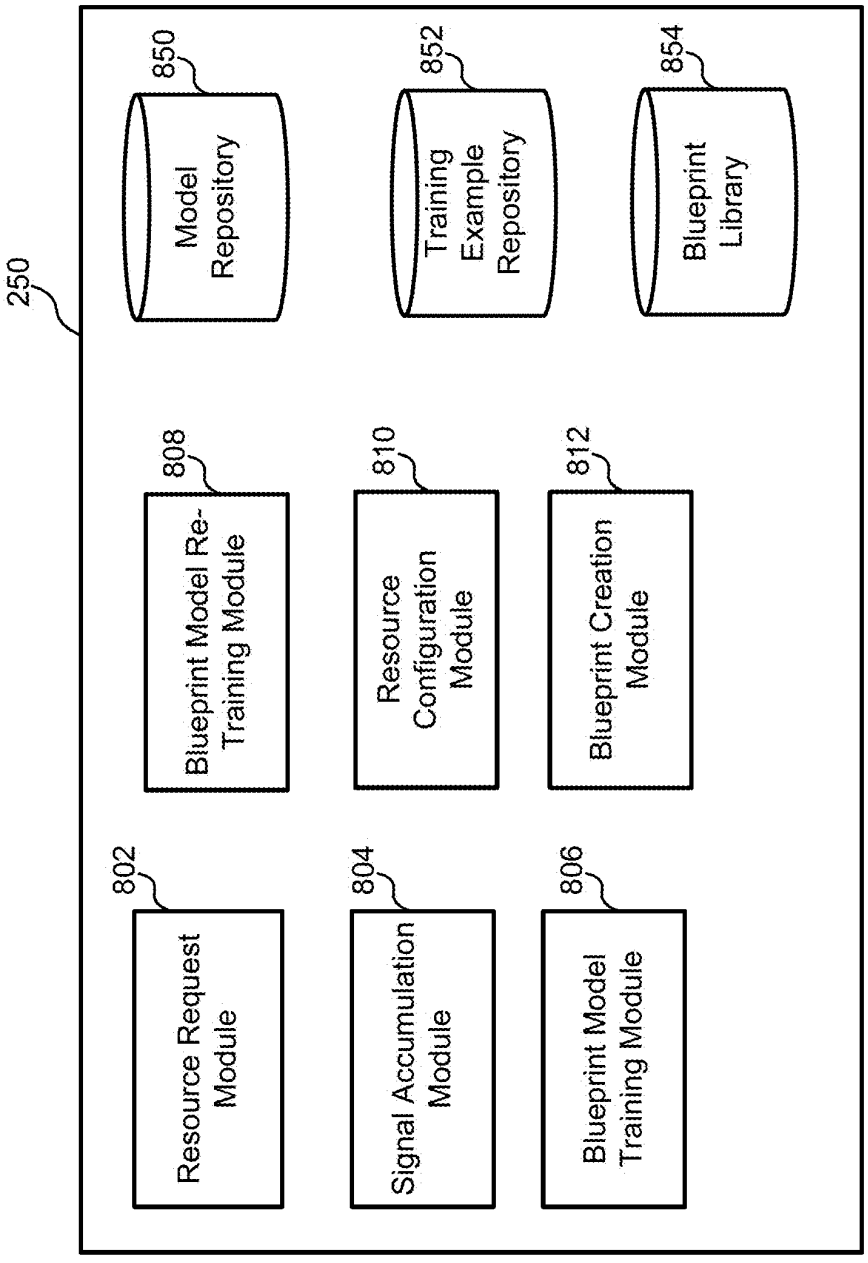

250
850
Model Repository
852
Training Example Repository
854
Blueprint Library
808
Blueprint Model Re-Training Module
810
Resource Configuration Module
812
Blueprint Creation Module
802
Resource Request Module
804
Signal Accumulation Module
806
Blueprint Model Training Module

FIG. 8

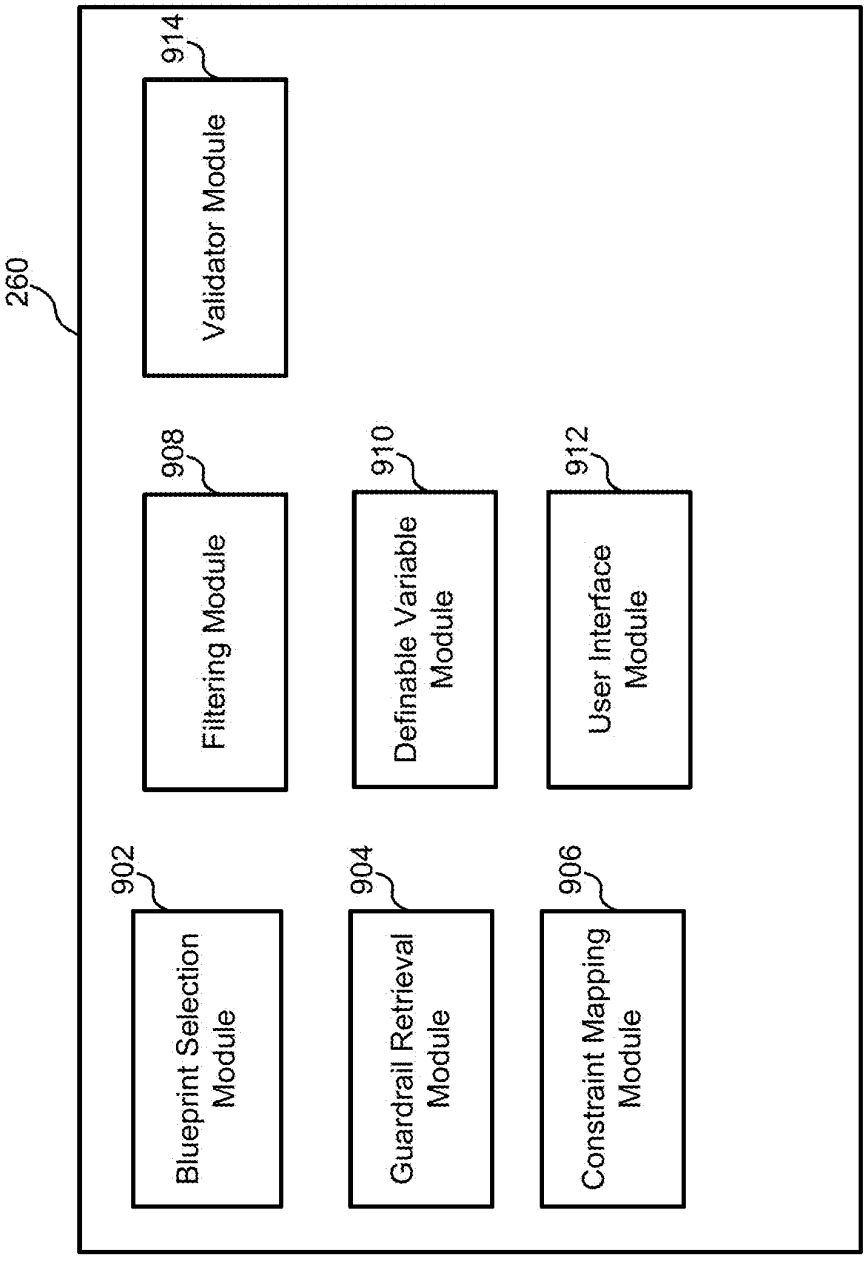

260
914
Validator Module
908
Filtering Module
910
Definable Variable Module
912
User Interface Module
902
Blueprint Selection Module
904
Guardrail Retrieval Module
906
Constraint Mapping Module

FIG. 9

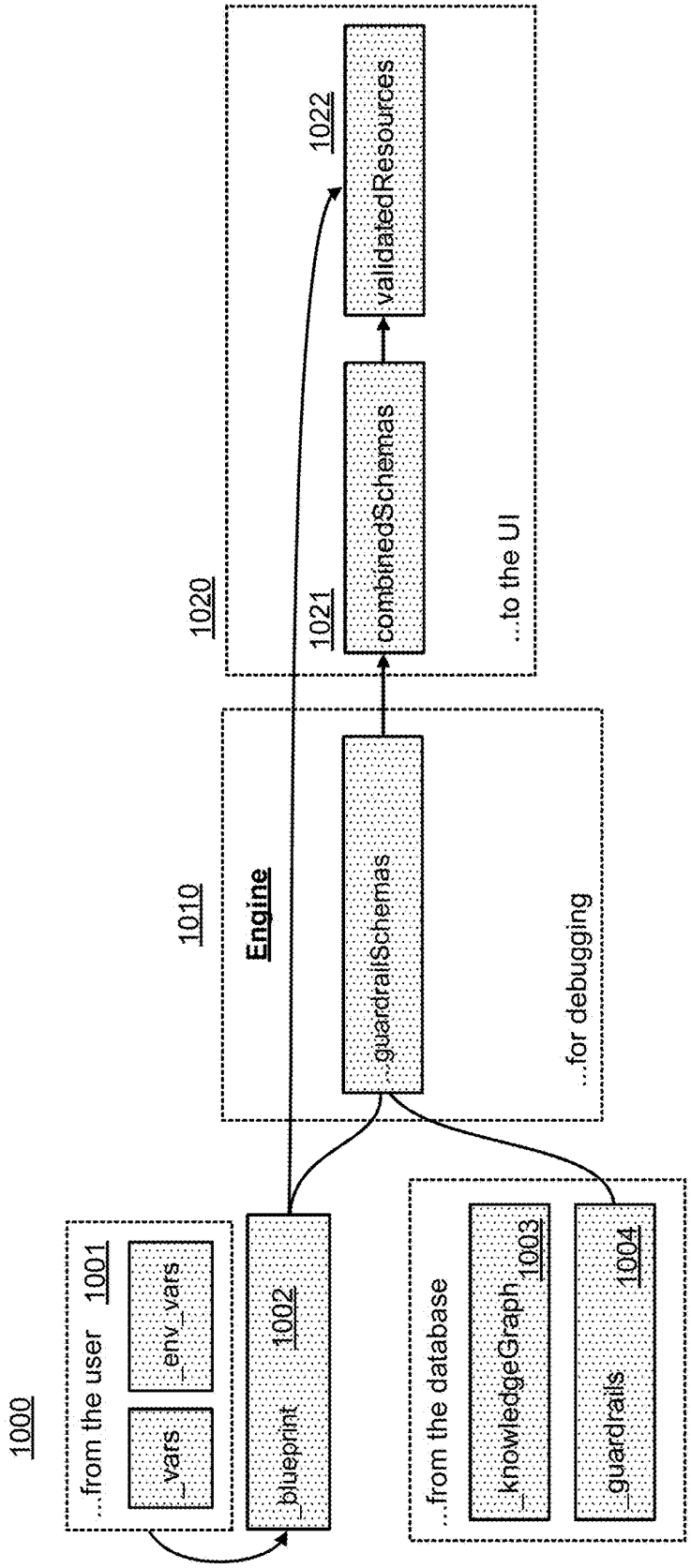

1000
...from the user 1001
vars
env_vars
blueprint 1002
...from the database
knowledgeGraph 1003
guardrails 1004
1010
Engine
guardrailSchemas
...for debugging
1020
1021
combinedSchemas
1022
validatedResources
...to the UI

FIG. 10

1101

```
---
constants:
  __name: "{{ bucket }}_{{ __guid }}"
---

resource "aws_s3_bucket" "{{ __name }}" {
  bucket = "{{ bucket }}"
}

resource "aws_s3_bucket_public_access_block" "{{ __name }}" {
    bucket = aws_s3_bucket.{{ __name }}.id

    block_public_acls       = true
    block_public_policy     = true
    ignore_public_acls      = true
    restrict_public_buckets = true
}

resource "aws_s3_bucket_ownership_controls" "{{ __name }}" {
  bucket = aws_s3_bucket.{{ __name }}.id

  rule {
    object_ownership = "BucketOwnerEnforced"
  }
}

resource "aws_s3_bucket_versioning" "{{ __name }}" {
    bucket = aws_s3_bucket.{{ __name }}.id
    versioning_configuration {
        status = "{{ versioning_configuration_status }}"
    }
}
```

FIG. 11

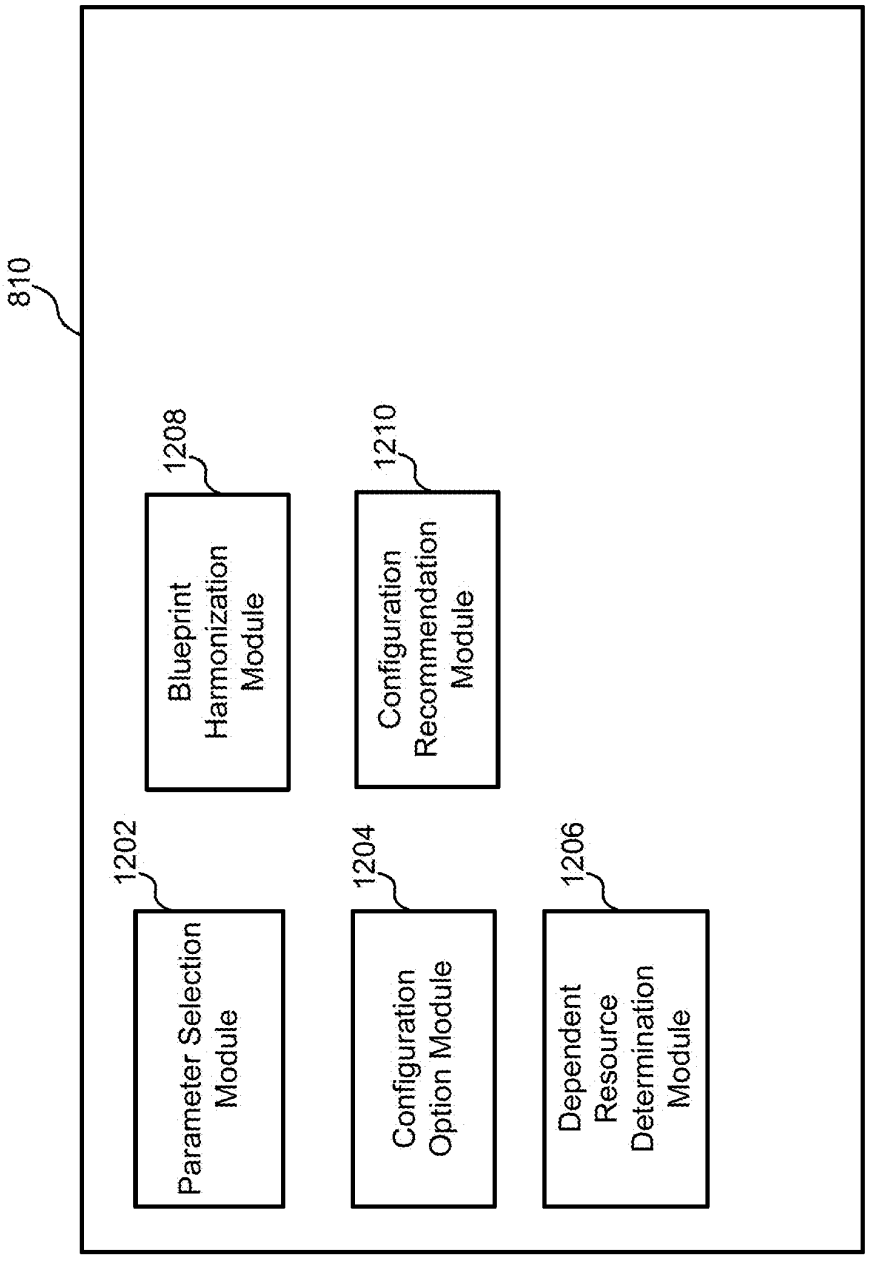

810
1208
Blueprint Harmonization Module
1210
Configuration Recommendation Module
1202
Parameter Selection Module
1204
Configuration Option Module
1206
Dependent Resource Determination Module

FIG. 12

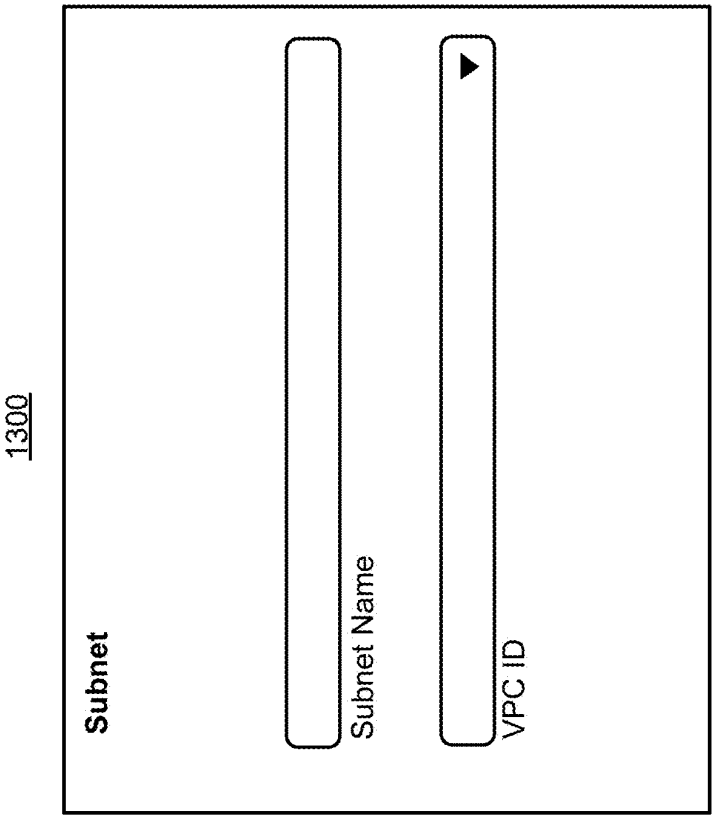


FIG. 13

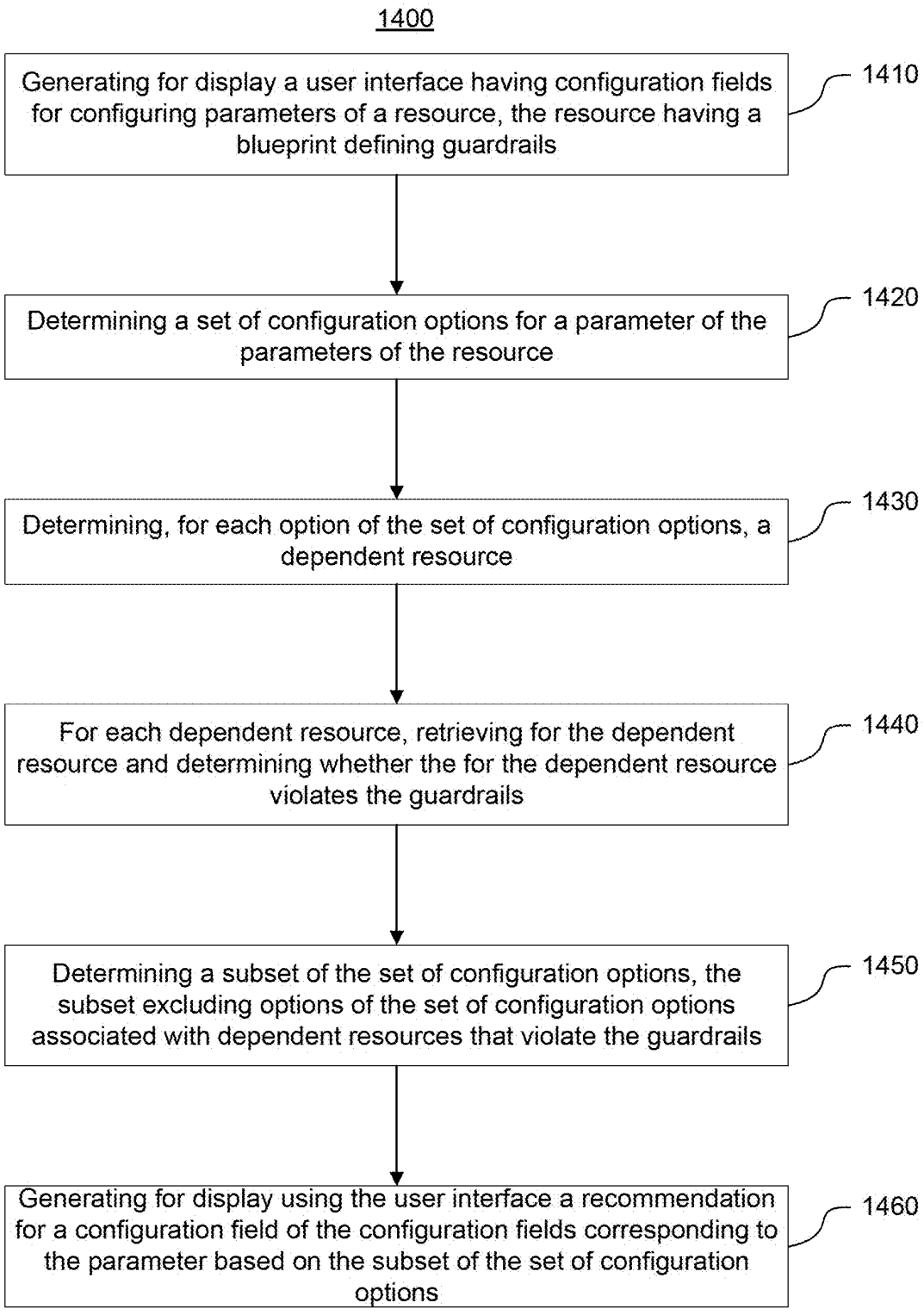

1400
Generating for display a user interface having configuration fields for configuring parameters of a resource, the resource having a blueprint defining guardrails
1410
Determining a set of configuration options for a parameter of the parameters of the resource
1420
Determining, for each option of the set of configuration options, a dependent resource
1430
For each dependent resource, retrieving for the dependent resource and determining whether the for the dependent resource violates the guardrails
1440
Determining a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails
1450
Generating for display using the user interface a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset of the set of configuration options
1460

FIG. 14

RESOURCE PARAMETER RECOMMENDATION ENGINE BASED ON DEPENDENT RESOURCES AND GUARDRAIL CONFLICTS

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 63/567,848, filed Mar. 20, 2024, the disclosure of which is hereby incorporated by reference herein in its entirety.

TECHNICAL FIELD

[0002] The disclosure generally relates to the field of computing resource configuration, and more specifically relates to using an improved user interface for parameter configuration that ensures downstream resources for candidate parameter configurations comply with resource configuration policy constraints.

BACKGROUND

[0003] In a typical resource development scenario, a developer of a service creates one or more computing resources. An operations team of the service then reviews the computing resource and determines whether it complies with operations policies (e.g., the resource is deployed in an authorized geographical location; the resource has security set up based on a policy governing personal identifying information (PII) where the resource contains PII, etc.). A security service then scans the resource for vulnerabilities and policy breaches and identifies security issues to the developers. These issues bottleneck developer progress, and add network and compute bloat in security scans and unnecessary communications between the parties that, at scale, can create massive inefficiencies for services.

[0004] Guardrails can be put in place to limit a universe of options that a user has when defining a resource, thereby removing this burden from other teams. However, when resources are chained together (e.g., where a parameter selection of one resource invokes use of a dependent resource), a conflict between guardrails for the two resources may violate a blueprint and cause the resource configurations to be non-compliant.

SUMMARY

[0005] Systems and methods are disclosed herein for an improved user interface that generates recommendations for parameter configuration options to users. The recommendations are based on a knowledge graph and are additionally based on a determination of resources from which different parameter configuration options depend. For example, where a resource being configured is a virtual machine (VM), a parameter that is to be configured may include a Virtual Private Cloud (VPC) to which the VM is to be deployed. Some VPCs may themselves have attributes that violate a guardrail of the VM's blueprint, and therefore are unusable. The recommendation engine factors in dependent resources from different possible configuration parameters in order to provide recommendations that are compliant with the resource's blueprint.

[0006] In some embodiments, a policy enforcement service displays a user interface having configuration fields for configuring parameters of a resource, the resource having a blueprint defining guardrails. The policy enforcement service determines a set of configuration options for a parameter of the parameters of the resource, and determines, for each option of the set of configuration options, a dependent resource. For each dependent resource, the policy enforcement service retrieves a blueprint for the dependent resource and determining whether the blueprint for the dependent resource violates the guardrails. The policy enforcement service determines a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails, and displays a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset.

BRIEF DESCRIPTION OF DRAWINGS

[0007] The disclosed embodiments have other advantages and features which will be more readily apparent from the detailed description, the appended claims, and the accompanying figures (or drawings). A brief introduction of the figures is below.

[0008] FIG. **1** illustrates one embodiment of a system environment for implementing a policy enforcement service.

[0009] FIG. **2** illustrates one embodiment of modules of the policy enforcement service.

[0010] FIG. **3** is a block diagram illustrating components of an example machine able to read instructions from a machine-readable medium and execute them in a processor (or controller).

[0011] FIG. **4** illustrates an exemplary user interface for defining a resource policy, in accordance with an embodiment.

[0012] FIG. **5** illustrates an exemplary user interface for creating a resource, in accordance with an embodiment.

[0013] FIG. **6** illustrates an exemplary user interface for selecting a blueprint for creating a resource, in accordance with an embodiment.

[0014] FIG. **7** illustrates an exemplary user interface for configuring a resource using a selected blueprint.

[0015] FIG. **8** illustrates one embodiment of sub-modules of a blueprint selection module.

[0016] FIG. **9** illustrates one embodiment of sub-modules of a configuration validator module.

[0017] FIG. **10** depicts a data flow for use in using an engine to drive a user interface for generating a new resource that conforms to existing policy.

[0018] FIG. **11** depicts exemplary user inputs for use with respect to a resource creation engine, in accordance with some embodiments of the disclosure.

[0019] FIG. **12** illustrates one embodiment of sub-modules of a resource configuration module, in accordance with some embodiments of the disclosure.

[0020] FIG. **13** illustrates an exemplary user interface for configuring a resource parameter associated with a downstream resource.

[0021] FIG. **14** illustrates a flowchart showing an exemplary process for generating parameter configuration recommendations based on dependent resource conflicts with a resource blueprint, in accordance with an embodiment.

DETAILED DESCRIPTION

[0022] The Figures (FIGS.) and the following description relate to preferred embodiments by way of illustration only.

It should be noted that from the following discussion, alternative embodiments of the structures and methods disclosed herein will be readily recognized as viable alternatives that may be employed without departing from the principles of what is claimed.

[0023] Reference will now be made in detail to several embodiments, examples of which are illustrated in the accompanying figures. It is noted that wherever practicable similar or like reference numbers may be used in the figures and may indicate similar or like functionality. The figures depict embodiments of the disclosed system (or method) for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles described herein.

[0024] FIG. **1** illustrates one embodiment of a system environment for implementing a policy enforcement service. As depicted in FIG. **1**, policy enforcement service environment **100** includes various client devices, including a developer device **110**, operations device **112**, and security device **113**, as well as network **120**, policy enforcement service **130**, and generative artificial intelligence (AI) tool **140**. While policy enforcement application **111** is only depicted with respect to developer device **110**, this is for convenience only, and may exist on any client device. Developer device **110** is operated by a developer of a resource. The term developer, as used herein, may refer to anyone who creates a resource, such as a developer, a person who launches cloud resources, or any user of a client device who creates resources. Operations device **112** is operated by a person dedicated to operations, such as compliance management, in association with a same service as the developer. Security device **113** is operated by a person dedicated to security compliance, and may be a third party relative to the service or may be part of the service. A service, as used herein, is a collection of one or more cloud resources that together, perhaps in coordination with other activities, form a client-facing tool.

[0025] Policy enforcement service **130** is used by client devices (e.g., operations device **112** and/or security device **113**) to generate guardrails. The term guardrail, as used herein, may refer to properties of resources that are to be adhered to by developers. The guardrails may be specific to types of resources—that is, databases having sensitive information is one type of resource, and databases accessible to certain geographies is another type of resource. The guardrails are defined by client devices having permissions to define constraints for given types of resources.

[0026] Guardrails may be established for resources of any kind, including newly created resources and existing resources that are imported. While newly created resources can be created in any manner (e.g., coded from scratch), blueprints may be used to create resources as well. For example, a library of blueprints may be available for quick resource creation, where a user can access the library and select a blueprint to create a resource. Guardrails may be established for new blueprints that are used to create resources, such that the blueprints, when used to create a resource, result in the resource adhering to the guardrails. Where blueprints are imported, guardrails may be imposed on those blueprints that, where blueprints are non-compliant, cause the blueprints to be modified in order to be usable following importation. Blueprints are described in further detail below.

[0027] After the guardrails are established, when developer device **110** generates a resource, policy enforcement application **111** forces the resource to have properties that adhere to the defined constraints. Policy enforcement service **130** is instantiated on one or more servers outside of the service of developer device **110**, accessible by way of network **120**. Policy enforcement application **111** is an application installed on developer device **110** and/or accessible by way of a browser of developer device **110**. Some or all functionality of policy enforcement service **130** described herein may be distributed or fully performed by policy enforcement application **111** on a client device, or vice versa. Where reference is made herein to activity performed by policy enforcement application **111**, it equally applies that policy enforcement service **130** may perform that activity off of the client device, and vice versa. Further details about the operation of policy enforcement service **130** are described below with reference to FIG. **2**.

[0028] FIG. **2** illustrates one embodiment of modules of the policy enforcement service. As depicted in FIG. **2**, policy enforcement service **130** includes policy definition module **202**, resource importation module **204**, reconciliation module **206**, owner determination module **208**, configuration recommendation module **210**, policy constraints database **212**, blueprint selection module **250**, and configuration validator module **260**. These modules and databases are merely illustrative; fewer or more modules and/or databases may be used to achieve the functionality disclosed herein.

[0029] Policy definition module **202** defines policies that apply to newly created resources, as well as apply to existing, imported resources in bringing those imported resources into compliance. The term policy, as used herein, may refer to a collection of guardrails that are applied to a resource or a collection of resources. A policy may be created on a per-resource basis, on a resource attributes basis (e.g., where a resource has one or more certain attributes, a policy applies), or any other basis. A resource may be subject to compliance with multiple policies.

[0030] A policy may be defined by any client device, but generally is defined by an operations device **112** and/or a security device **113**. In an embodiment, policy definition module **202** receives code lines from a client device that define a policy. In another embodiment, policy definition module **202** receives input of functional blocks from a client device (e.g., by way of a user interface of policy enforcement application **111**) that define a policy. For example, turning to FIG. **4**, user interface **400** includes a defined policy, where if a data classification label contains personal identifying information relating to Europe, and if a resource environment relates to a product, then the resource must be created in a geographical location defined as Europe West 1. In an embodiment, policy definition module **202** detects a selection of each segment of the policy (e.g., each "if" or "require" bar, by way of clicking, touching, drag-and-drop, or any other method of selection) by way of user input into a graphical user interface of a client device. Following implementation of this policy, newly created resources having these attributes will be forced to be created in Europe West 1 (e.g., by only offering Europe West 1 as an option for location). An imported resource that has these attributes but is located in a different location (e.g., Europe East 2) would be forced to migrate to Europe West 1.

[0031] Policy definition module **202** may display candidate segments in a menu, list, or other navigable tool for

selection, where users may select from functional blocks including conditions (e.g., "if" statements), as well as requirements (e.g., what to do where conditions are met). This enables users who are not fluent in drafting computer code to nonetheless develop policies that developers are to adhere to. In an embodiment, policy definition module **202** outputs recommendations of one or more individual candidate segments and/or recommendations of candidate collections of segments that together can form a policy. Policy definition module **202** may train a machine learning model using training examples to generate recommendations. The training examples may be specific to a user based on prior policies created by the user, or may be specific to a group of users (e.g., training examples across a team, department, or conglomerate may be used). The training examples may include collections of segments as labeled by a resource and/or resource type and/or resource attribute.

[0032] Policy definition module **202** may input data into a machine learning model as a user supplies information. The supplied information may be a selection of one or more segments and/or other information about the resource to which the policy will apply. Policy definition module **202** may receive as output from the machine learning model probabilities that different candidate segments would be selected and/or different collections of candidate segments that may apply. Policy definition module **202** may output on the user interface a ranked list of candidate segments, the rankings being based on the probabilities. The ranked list may be truncated to at most include a predefined amount of segments, and/or may be truncated to include candidate segments that have at least a threshold probability of being selected. Policy definition module may receive a selection from the user, and may re-train the machine learning model using that feedback. This may result in a different ordering of candidate segments in the future. For example, where the same input is provided by the user in the future, a different ordering of candidate segments may be shown to the user in the ranked list based on the prior application usage by the user. Where all recommendations are ignored and the user selects different candidate segments that were not part of the recommendation, the machine learning model may be retrained to reflect negative bias toward each candidate segment with respect to the inputs provided by the user.

[0033] As policies are defined, policy definition module **202** populates policy constraints database **212** with the policies. From here, as developer device **110** creates resources and defines attributes of those resources, policy constraints database **212** is queried for guardrails relating to those attributes. As an example, turning briefly to FIG. **5**, user interface **500** shows a user creating a resource (e.g., a bucket or data store) that will hold PII of a European Union citizen. The guardrail from the policy established in FIG. **3** exists, and so responsively, policy enforcement service **130** references policy constraints database **212** and determines that the EU-west-1 region should be offered based on the policy. Additionally, policy enforcement service offers an "other" option as a guardrail, where any other region for deployment requires approval (e.g., by an entity defined in the policy, or by a default approver such as a user of an operations device **112** or security device **113**).

[0034] Returning to FIG. **2**, resource importation module **204** imports existing resources into an environment of policy enforcement service **130**. For example, a service may have existed outside of environment **100** and may elect to join environment **100**. In such a case, newly created resources would conform to policies for that service, but older, existing policies may need to be reconciled in order to conform to those policies. Reconciliation module **206** detects the importation of a new resource, and determines whether a configuration of that resource is not in compliance with policy constraints **212** that relate to that resource. To this end, reconciliation module **206** determines a type of the pre-existing resource, and queries policy constraints **212** to determine policies that apply to a resource of that type (e.g., a data store or bucket as in the example of FIG. **4**). The term type, as used herein, may refer to a category of resource. The category of resource may reflect one or more attributes of the resource. A resource may have more than one type. As an example, a resource for hosting personally identifying data may be of the type "storage" and also of the type "sensitive." A data structure that maps attributes to types may be referenced to determine a type of a resource. Responsive to determining that the pre-existing resource is of the given type, reconciliation module **206** determines whether the pre-existing resource does not comply with the policy constraints, and if so, begins a reconciliation process. To determine whether a pre-existing resource complies with policy constraints, reconciliation module **206** may compare attributes of the pre-existing resource to guardrails of a policy. Compliance does not occur where a guardrail would have been broken had the pre-existing resource been created to conform to the policy, rather than having been created elsewhere and then imported.

[0035] A reconciliation process may begin with reconciliation module **206** determining an owner of a resource. The term owner, as used herein, may refer to an individual person, a group of individuals, or a person having a certain credential (e.g., a vice president or higher level employee in a certain division is defined to be an owner). In some embodiments, a resource may have edit protections where only an owner of the resource can edit the resource. Metadata of the resource may indicate the requirements of who qualifies as an owner. However, in some cases, the owner may not be defined by the resource. To this end, owner determination module **208** may use metadata of the resource to determine a log source of the resource (e.g., an event log relating to creation or maintenance of the resource, such as a CloudTrail log or a Git history for resources that were checked in to infrastructure-as-code repositories). Ownership determination module **208** may identify a user identifier within a log of the log source (e.g., a handle or contact address of a candidate owner), and may determine that the owner is the person identified by the user identifier. In an embodiment, owner determination module may prompt the owner to confirm that that user is indeed the owner. In another embodiment, owner determination module **208** may simply conclude that this person is the owner.

[0036] In an embodiment, ownership determination module **208** accesses a machine learning model trained to identify an owner of the file. Ownership determination module **208** may input code lines of the file and/or metadata of the file into the machine learning model. The machine learning model may output a prediction of who the owner of the file is. The output may be a direct prediction, or may assign probabilities to each user identifier named within the resource as to whether that user identifier is or is not an owner. Where probabilities are assigned, ownership determination module **208** may determine that a user identifier

corresponds to an owner responsive to determining that the probability output by the machine learning model exceeds a threshold value.

[0037] The machine learning model may be trained by generating embeddings for different segments of code and metadata within a resource. For example, lines of code and metadata that include a user identifier may be converted into a semantic representation in latent space using a supervised machine learning model. Owner determination module **208** may then use an unsupervised machine learning model to determine the distance in latent space between one or more example owner embedding representations and each semantic representation. Where a distance is below a threshold, owner determination module **208** may determine that the user identifier within the corresponding text to the latent representation is the owner.

[0038] After the owner is determined, configuration recommendation module **210** prompts the owner with a set of recommended configuration changes, which may be determined based on a comparison of configuration settings of the pre-existing resource to the policy constraints. For example, if a resource includes PII and European Union citizen data but is stored somewhere other than the EU-west-1 region, then the recommended configuration changes may include relocating the resource to the EU-west-1 region. As another example, a policy constraint may indicate that a resource should only interact with a predefined number (e.g., 1) of applications, and reconciliation module **206** may determine that the resource interacts with 4 applications. Configuration recommendation module **210** may prompt the owner to remove access to 3 of those applications in this example. Moreover, configuration recommendation module **210** may recommend for which application(s) to maintain access (e.g., based on ranking the applications on some metric, such as access frequency).

[0039] Configuration recommendation module **210** may recommend changes using a machine learning model. Configuration recommendation module **210** may input into the machine learning model the policy constraints and attributes of the resource (or a portion thereof, such as attributes not in compliance with the constraints), and may receive as output from the machine learning model a recommendation of what to change in the attributes of the resource to comply with the constraints. The machine learning model may be trained using historical data, where each training example in the historical data includes resource attributes and constraints as labeled with changes to the resource attributes that were taken. Configuration recommendation module **210** may output for display the recommendation of what to change to the owner.

[0040] These recommendations may be selectable and, responsive to receiving a selection of a selectable option from the owner, configuration recommendation module **210** may reconfigure the resource with the recommended configuration changes. In some embodiments, responsive to receiving a selection of a selectable option from the owner, configuration recommendation module **210** may retrain the machine learning model that is used to output a recommendation to add a positive bias toward the selected option and/or a negative bias toward unselected options, thereby resulting in a change in recommendations in the future.

[0041] In an embodiment, configuration recommendation module **210** may open a pull resource to change an infrastructure-as-code configuration of the resource and prompt the owner to apply the change. In either case, reconfiguration and/or application may involve restructuring the resource itself. For example, where a data store is moving regions, configuration recommendation module **210** may generate a new resource in the region to which a resource is to be moved, command that data be copied from the old region to the new region (e.g., from EU-east-1 to EU-west-1), and then command that the old resource be torn down. Tear downs and migrations may occur in similar scenarios, such as where a policy requires migration to a server having more security from a server having insufficient security. In such cases, configuration recommendation module **210** may perform the requisite restructuring conforming to the selection of the owner, including teardowns and migrations.

[0042] Blueprint selection module **250** drives a user experience (UX) to guide a user in creating a resource. The term blueprint, as used herein, may refer to a framework for creating a resource. The framework may include pre-configured parameters, such as policy constraints that must be adhered to when creating the resource. Policy constraints (e.g., guardrails) may be part of policies generated in any manner described above with respect to FIGS. **2-5**, and may automatically be applied to a blueprint based on a type of resource that can be generated by the blueprint. For example, where a blueprint is for creating a resource that stores personal identifying information and European Union citizen data, the blueprint may only offer to set up the resource in Europe West 1, consistent with the above examples used with respect to FIGS. **2-5**.

[0043] The framework may also include a set of fields where input is required that together form the information needed to create the resource. Policies and policy constraints for a given blueprint may be defined in any given manner described above (e.g., with respect to policy definition module **202**), where, rather than expressly defining a policy for a single resource, the policy is instead defined for any resource created using a given blueprint. Blueprints may be newly created, or may be imported. Where blueprints are imported, all activities described with respect to resources in connection with resource importation module **204**, reconciliation module **206**, owner determination module **208**, and configuration recommendation module **210** equally apply. That is, when a blueprint is imported, the imported blueprint may be checked to ensure that resources created using that blueprint comply with existing policy constraints for that type of resource.

[0044] Turning briefly to FIG. **6** for illustrative purposes, FIG. **6** illustrates an exemplary user interface for selecting a blueprint for creating a resource, in accordance with an embodiment. As shown in FIG. **6**, a plurality of (in this case, three) different blueprint options are available from user interface **600**—a web services bucket (named AWS bucket) for private and intranet data storage, a cloud object store (named GCP storage) for highly sensitive data, and a web services bucket (named AWS Bucket as well) for public data. Different guardrails may apply for the different types of data storage based on their associated policies. User interface **600** may include any number of blueprints for selection. In an embodiment, all blueprints may be presented in user interface **600**. In an embodiment, the blueprints shown in user interface **600** may be results from a search (e.g., in this example, where a user is searching for a blueprint for creating a storage resource). As described below, the blueprint options may be ranked and ordered based on a recom-

mendation engine driven by blueprint selection module **250**, and application usage by users in selecting and using blueprints to create resources may change how the blueprint options are ranked and ordered over time, thus resulting in an improved user interface. Blueprint options are selectable options that, when selected from user interface **600**, result in configuration prompts for the user as shown in FIG. **7**.

[0045] Turning next to FIG. **7** for further illustration, FIG. **7** illustrates an exemplary user interface for configuring a resource using a selected blueprint. As depicted in user interface **700**, after a blueprint is selected, configuration prompts are generated for display to the user. At least some of the configuration fields are custom built for the selected blueprint. For example, a user might be required by the blueprint to indicate what team manages data within the resource, but other blueprints may not require this data. The blueprint may have preconfigured policy constraints—for example, out of five possible environments for the resource, the policy constraint may only make two of those possible environments available where this blueprint is used. These policy constraints may be associated with their corresponding blueprints using a data structure within policy constraints database **212**. User interfaces **600** and **700** may be part of policy enforcement application **111** and may be driven as part of processes of resource configuration module **810**, described further below with reference to FIG. **8**.

[0046] FIG. **8** illustrates one embodiment of modules of the blueprint selection module. A description of modules of the blueprint selection module **250** that act to perform functionality that enables the UX described above with respect to FIGS. **6-7** follows. As depicted in FIG. **8**, blueprint selection module **250** includes resource request module **802**, signal accumulation module **804**, blueprint model training module **806**, blueprint model re-training module **808**, resource configuration module **810**, and blueprint creation module **812**. Blueprint selection module **250** also has access to model repository **850**, training example repository **852**, and blueprint library **854**. These modules and databases are merely illustrative; fewer or more modules and/or databases may be used to achieve the functionality disclosed herein.

[0047] Resource request module **802** may receive, based on input by a user, a request to generate a resource. Resource request module **802** may receive the request based on a selection of an icon of policy enforcement application **111** that corresponds with resource creation, or an express selection of an icon to access candidate blueprints, or through any other command aimed at achieving generation of a new resource. The request may include one or more search parameters that cause a results list of matching blueprints to be displayed in the UX.

[0048] In an embodiment, resource request module **802** causes display within the UX of a field that accepts one or more search parameters. The search parameters may include descriptors of relevant resources, such as any combination of resource title, resource type, resource attribute(s), resource requirements (e.g., must be deployable in EU East 1), and/or any other parameter useful in locating candidate blueprints (e.g., recency parameters, size parameters, and so on). In response to receiving the request, resource request module **802** may search for blueprints matching the parameters within blueprint library **854**. Blueprint library **854** may be a repository of blueprints, which may be indexed according to any number of searchable parameters.

[0049] Prior to causing a display of the user interface having the plurality of blueprints for selection, in some embodiments, blueprint selection module **250** determines which blueprint icons to display, and in what order to display them. To this end, blueprint selection module **250** may input signals (accumulated by signal accumulation module **804**) into a machine learning model (stored in model repository **850**), and may receive output of blueprints corresponding to the signals, from which blueprint selection module **250** may determine which blueprint icons to display. The supervised machine learning model may be, for example, a deep neural network, a convolutional neural network, or any other supervised machine learning model, as stored in model repository **850**.

[0050] The machine learning model may be trained by blueprint model training module **806** using training examples stored in training example repository **852**. For example, each blueprint may have a plurality of attributes (e.g., what the resource that it will generate is, how the resource it will generate is configured, policies for the resource to be generated, policy constraints for the resource to be generated, and so on). The supervised machine learning model may be trained using training examples having sets of signals that are labeled. The signals may correspond to one or more of a profile of an entity that selects the blueprint. The entity may include one or more of a individual, a team on which the individual is place, a domain associated with the individual, a classification of the domain, and so on. For example, data from a profile of a user that selects the blueprint may form part of the training data, including information about the user and information about activities of the user (e.g., blueprint options that the user did or did not select and context information surrounding each of those selections, such as information about other resources created by the user within a time interval of making that selection). Profile data may include any other information known about a user or other entity, such as characteristics of the user/users within an entity, and so on.

[0051] Similarly, data from profiles of teams (e.g., profiles of multiple users that form a team) may be taken in the aggregate as signals. Data of domain (e.g., a domain in which the team operates where there are multiple teams within that domain) may be taken from profiles of the users within that domain in similar fashion, and so on. A classification of the domain may be used as a signal (e.g., a resource is being created for a domain in the information technology space versus the administrative space, information technology and administrative being example classifications). The signals may also include search parameters received using resource request module **802**.

[0052] Regardless of the set of signals for each training example, the training example may be labeled. The labels may indicate whether a blueprint having a given set of attributes (e.g., attributes of the blueprint that may form part of the training data) presented to an entity having their own given profile attributes was selected by the entity, and may also indicate context data, such as other blueprints that were and were not selected by the entity for creating a given resource. For example, it is informative not just whether a given blueprint is selected, but which blueprints were not selected in a given scenario, as the model can then be trained to predict, given a set of candidate blueprints having respective attributes, a likelihood that a given user would select a

given one of the set of candidate blueprints. The training examples may be stored in training example repository **852**.

[0053] Blueprint model training module **806** may train one or more machine learning models using the training examples stored in training example repository **852**, yielding models stored in model repository **850**. In order to train one or more models, blueprint model training module **806** may retrieve training examples that conform to a policy (e.g., train a model for a specific user, for a specific team, for a specific domain, for an entire entity, and so on), and may train a model using the conforming training examples. Models may be trained on-the-fly as a given user requests creation of a resource according to a policy dictating how to train the model for that user. In some embodiments, models may be trained in advance and may be retrieved from model repository **850** for usage responsive to resource creation requests being received, the model being retrieved according to who the user is and a policy dictating which model is to be used. Machine learning models may be generic and operate across different teams within a given domain. Alternatively or additionally, given domains and/or teams and/or users may each have their own machine learning models that are trained specifically for that domain. For example, where training data is sensitive and activity of a domain is to remain secure, training data may not be permissible to be used to train a model to be used outside of that domain, and thus a domain-specific model may be used.

[0054] Following training one or more models, each model is equipped to take as input a set of signals and, given a set of candidate blueprints, rank and order the candidate blueprints in terms of likelihood that a given user would select each given candidate blueprint. This may be performed with signal accumulation module **804** determining a set of signals to input into a supervised machine learning model. To this end, the signals may include any combination of data about a user, a team on which the user operates, a domain of the user, a classification of the domain of the user, and so on. As a proxy for this data, the signals may include an aggregate set of signals for users like the given user. Moreover, the signals may include contextual information, such as other resources recently created by the user (e.g., because the model may be able to predict that where a given resource is created, a next resource is likely to be created). The signals may include profile data of the user (or users within a team, domain, and so on). The signals may include search parameters input by the user and other parameters derived from those search parameters (e.g., synonyms). Based on the set of signals, the machine learning model may output a probability for each given blueprint of the plurality of blueprints that the user will access the given blueprint. The blueprints evaluated may be blueprints that are available to the user for use by a domain of the user, and may be further limited by search terms of the user for blueprints that pertain to resource creation that satisfies certain specified criteria.

[0055] Blueprint selection module **250** may assign ranks to each given blueprint of the plurality of blueprints based on their corresponding probabilities, and may order the plurality of blueprints based on the ranks. Using the order, blueprint selection module **250** may generate for display user interface (e.g., user interface **600**), each blueprint comprising a selectable option that, when selected, leads to fields for configuring the resource. That is, blueprint options in user interface **600** may be ordered based on a likelihood that the user will use each given blueprint for creation of the resource.

[0056] In an embodiment, responsive to a blueprint being selected, blueprint model re-training module **808** may cause the supervised machine learning model to be re-trained. Such retraining may be based on which blueprint of the plurality of blueprints is selected. For example, where a highest-ranking blueprint is selected, biases that led to the highest-ranking blueprint to be ranked first may be strengthened. Where a lower ranking blueprint is selected, its biases may be improved, and other biases for the other candidate blueprints may be weakened. This may result in a scenario where assigning ranks and ordering a future a set of blueprints in a future request to generate a resource is altered based on the supervised machine learning model being re-trained. This results in an improved user interface that updates blueprint icon ordering for a recommendation to a user based on application usage.

[0057] In an embodiment, blueprint selection module **250** may use generative artificial intelligence (AI) based on a large language model to recommend one or more blueprints from blueprint library **854**. Generative AI poses challenges in computational time and expense, in that when it is prompted with a query, the models used explore a huge universe that requires immense processing power. In an embodiment, blueprint model training module **806** may prime a generative AI model with a limited context window to improve the computational efficiency by one or more orders of magnitude. Specifically, blueprint library **854** may be associated with a metadata catalog. The metadata catalog may include natural language that describes features of a blueprint, such as a title, a description, options available for configuring the resource, properties, and any other attributes.

[0058] As an example, a blueprint may be associated with building a "Back-up Bucket". The metadata catalog may include "Description: Bucket for backing up files, versioning is on and files will be rotated to Nearline/Coldline/Glacier after a configurable number of days." The metadata catalog may also include: "Options: (1) Bucket name; (2) Location (if applicable); (3) Number of days before moving to Nearline storage (if applicable); (4) number of days before moving to Coldline storage (if applicable); (5) Age of an object before transitioning to Glacier (if applicable)." For properties, the metadata catalog may include "Properties: (1) Force destroy enabled; (2) Uniform bucket level access; (3) Public access block; (4) Incomplete multi-part uploads are aborted after 7 days." Blueprint library **854** may have similar associated catalog entries for any or all blueprints within the library.

[0059] Blueprint model training module **806** may prime a generative AI model by feeding it the metadata catalog for searchable context. This enables a user to enter a natural language query (e.g., using resource request module **802**), where the query is fed as input into the generative AI model, and where a search space performed by the generative AI model is informed by the context of the catalog, reducing search spaces in other areas and thereby reducing computational power required and time required to process the query. As an example, resource request module **802** may receive a query of "Recommend a blueprint for an S3 bucket." The generative AI model may search using the context of the catalog, and may provide one or more

candidate blueprints (in this case, one or more candidate blueprints for building a bucket). Where more than one candidate blueprint is surfaced, they may be ordered based on use of signals accumulated by signal accumulation model **804** as described in the foregoing, which may result in an ordering of candidate blueprints being made on the user interface using ranking as described above.

[0060] Resource configuration module **810** may be used to configure a resource in any manner discussed in the foregoing with respect to FIGS. **2-7**. Returning to FIG. **7**, in some embodiments, the same or a different supervised machine learning model may be used to suggest input into fields of user interface **700**. For example, the machine learning model may ingest some or all of the set of signals, and may determine likely inputs into any given field of user interface **700** for creating a resource using a selected blueprint from the recommended candidate blueprints. The machine learning model may output one or more suggestions (e.g., by pre-filling a given field with a tab-to-complete suggestion) for what to input into that field. For example, where a typical user having a profile of the user filling out user interface **700** typically inputs a certain string into a team field, the machine learning model may output a prediction that this string will be input there, and such a string may be suggested for input. Blueprint model re-training module **808** may re-train the model used to output the suggestion in a manner similar to re-training to output recommended blueprints based on whether each suggestion in accepted or declined.

[0061] In an embodiment, to enable seamless blueprint creation, guardrails may be used as building blocks for blueprint creation and may apply one-to-many with blueprints. Blueprint creation module **812** may enable users to create blueprints from scratch. In an embodiment, blueprint creation module **812** receives express input of guardrails for each blueprint. In another embodiment, guardrails may be defined to automatically apply to blueprints having certain attributes. For example, a guardrail may define that "all buckets having Property 1 must be deployed in Region 1". Thereafter, as blueprint creation module **812** receives input that a blueprint is being created for buckets having Property 1, blueprint creation module **812** determines that Region 1 for deployment based on the guardrail, and automatically assigns Region 1 for that blueprint. Blueprint creation module **812** may generate detection logic where as blueprints are created, conditions are monitored that are indicated in the detection logic. Responsive to detecting one of those conditions, blueprint creation module **812** applies the corresponding rule from the guardrail in which the condition is established.

[0062] Blueprints may be generated in a manner that makes them automatically modifiable as guardrails are updated. Blueprint creation module **812** may receive a command to implement one or more guardrails for a blueprint, where the guardrails define one or more tags. Tags may be mapped to a data structure that defines the guardrails, where the data structure is modifiable by one or more users. Following creation of a blueprint, where the data structure corresponding to tag is modified, that modification applies to the blueprints featuring the tag, thus causing the blueprints to include that update as new resources are created using that blueprint. Moreover, existing resources generated using blueprints may feature those tags as well, thus enabling automatic updating of those existing resources in the same manner. As an example, a guardrail may specify that Security Feature 1 must apply to buckets having Property 1. When Security Feature 1 is overwritten in the data structure corresponding to these buckets by Security Feature 2, all buckets having Property 1 automatically update to using Security Feature 2, and all blueprints for creating such buckets are automatically updated to feature a guardrail for Security Feature 2 as well.

[0063] FIG. **9** illustrates one embodiment of sub-modules of a configuration validator module. As depicted in FIG. **9**, configuration validator module **260** includes blueprint selection module **902**, guardrail retrieval module **904**, constraint mapping module **906**, filtering module **908**, definable variable module **910**, user interface module **912**, and validator module **914**. More or fewer modules may be used to achieve the functionality disclosed herein. Together, these modules may drive a configuration engine.

[0064] In a traditional system, an engine is provided with a schema as input. The schema may include constraints (e.g., guardrails) and a configuration. The engine is designed to act as a validator, where it determines whether the configuration complies with the constraints, and outputs a binary validation (e.g., outputs whether the configuration is valid against the constraints). The engine may also be designed to populate default values (e.g., where explicit values are not provided in a configuration of a schema).

[0065] There are limitations in use of such an engine. For example, such an engine does not provide information to a developer as to, if the configuration is invalid, how the configuration may be modified in order to comply with the constraints. Moreover, where a configuration is valid, a user is not provided with guidance on other allowable values for the configuration.

[0066] Existing engines may check each constraint against a configuration. If a constraint is violated, the engine may output an error message. Existing engines may also simplify constraints, by unifying constraints (<5 and <10 simplify to just <5). An exemplary flow may include a sequence of steps. For Step (1), for each constraint merge the constraint into the configuration (distribute it over the applicable portions of the config). If the configuration value satisfies the constraint, the merged value is the configuration value. If the configuration value does not satisfy the constraint, the merged value is an error. If the configuration value is incomplete (e.g., might satisfy the constraint when completed), the result may be an abstract expression combining the value and the constraint.

[0067] For step (2), after all the constraints have been merged, if any values are errors, the system may determine that the validation has failed. If any values are still incomplete, the system may prompt the user to provide more input. If all values are complete, the system may output that the configuration was valid. This approach throws away info that is needed to provide a good UX (user experience). For example, the constraints that are satisfied are lost (replaced by the valid value), so the UX cannot render those constraints. Moreover, constraints from all sources are merged, so the UX cannot inform the user which constraints come from which sources. Systems and methods disclosed herein improve over these UX limitations.

[0068] Systems and methods are disclosed herein for implementing an improved engine **1010** that drives an improved user interface for generating resources that comply with constraints. Some exemplary improvements may

include, where a configuration is valid, determinations may be made as to other allowable values for the configuration, and a user interface may be provided to the user as to those values. Moreover, where a configuration is invalid, the system may determine how to modify the configuration in order to reach a valid configuration, including information on valid values for the configuration, as well as guidance on which constraints to override or disable.

[0069] To drive the engine, blueprint selection module **902** selects a blueprint for configuration of a resource. In some embodiments, blueprint selection module **902** performs this selection by first receiving user input of a set of variables for the resource, and then selecting the blueprint from a plurality of candidate blueprints based on the user input, where the blueprint accommodates each variable of the set of variables. Turning briefly to FIG. **10** to explain user inputs of these variables, FIG. **10** depicts a data flow for use in using an engine to drive a user interface for generating a new resource that conforms to existing policy. As depicted in FIG. **10**, inputs **1000** are used as input to engine **1010**, which drives outputs **1020**. Inputs **1000** may include inputs which are received from a user, inputs **1002**, which may be received from a user or determined automatically by policy enforcement service **130** based on inputs **1001**, and inputs **1003** and **1004**, which may be retrieved from one or more databases (e.g., a knowledge graph, policy constraints database **212**, and so on).

[0070] Inputs **1001** may be received from a user, and may include variables such as regular variables for the resource to be created and environment variables. Regular variables are shared across all of the configurations of the scenario being configured (that is, regardless of which environment is being used). Environmental variables are environment specific (e.g., specific to the use of the configuration, such as specific to staging versus production). For example, turning briefly to FIG. **11**, FIG. **11** depicts exemplary user inputs for use with respect to a resource creation engine, in accordance with some embodiments of the disclosure. As shown on lines 6-7 of inputs **1101**, the input may indicate that a resource called "aws_s3_bucket" is of a type "bucket." Other lines may define attributes of the bucket (e.g., regular variables) and/or attributes of the environment (e.g., environmental variables), such as an indication of which policies (e.g., guardrails and blueprints) control deployment of the bucket on a per-environment basis.

[0071] Returning to FIG. **9**, blueprint selection module **902** may determine, based on the variables input by the user, which blueprint to use for the resource configuration. In some embodiments, policy enforcement service **130** may determine the blueprint using any methodology described with respect to blueprint selection module **250** (described with respect to, e.g., FIGS. **2** and **8**). For example, generative AI may be used, where inputs **1001** are transformed by blueprint selection module **902** into vector representations in latent space, and compared against vector representations of blueprints in a blueprint catalog, where a highest matching entry of the blueprint catalog may be selected and the corresponding blueprint may be selected. Any other means of blueprint selection may be performed. In some embodiments, a user may select the blueprint directly, and this may be performed in addition to or in lieu of obtaining inputs **1001** from the user.

[0072] In some embodiments, blueprint selection module **902** selects the blueprint comprises by first identifying a subset of the plurality of candidate blueprints, each candidate blueprint of the subset accommodating each variable of the set of variables. That is, blueprint selection module **902** may perform a search to blueprint library **854** with the variables, where blueprint library **854** returns a subset of its full set of candidate blueprints, where the subset blueprints each allow for each of the variables to form part of a configuration. Blueprint selection module **902** may then inputting the subset of the plurality of candidate blueprints and profile information of a user into a machine learning model and receive, as output from the machine learning model, a recommendation of the blueprint. This forms a recommendation tailored to a user of a blueprint, and is described in further detail with respect to FIG. **8**.

[0073] Guardrail retrieval module **904** retrieves guardrails corresponding to the resource. Guardrails may be retrieved using any manner disclosed in the foregoing, such as retrieving guardrails that map to the selected blueprint from policy constraints database **212**, or in any other manner.

[0074] Constraint mapping module **906** inputs the blueprint and the guardrails into an engine, and receives, as output from the engine, a mapping of constraints to respective variables of the blueprint. The mapping of constraints is a mapping of, for each variable, what allowable values can be used out of the universe of possible values. For example, where a guardrail exists that requires PII to be stored in Europe West 1, the value of a location variable is mapped to a constraint of Europe West 1.

[0075] Turning back to FIG. **10** to further explain the activity of the engine, engine **1010** receives inputs **1002** (that is, one or more blueprints) and inputs **1004** (that is, guardrails for an enterprise), and outputs guardrail schemas that are applicable given inputs **1001** and blueprints **1002**. The guardrail schemas form a compilation of constraints for the variables called for by the blueprint, as constrained by the guardrails. In practice, users may be generating configuration for multiple environments at once (e.g., deploying resources to development, staging, and production environments). In these scenarios, the configurations may be nearly identical, with just some differing values (IP ranges, DNS names, instance counts, instance sizes, etc.). Engine **1010** may include special support for these scenarios. To support these scenarios, engine **1010** may take inputs of multiple types including both regular variables and environmental variables, where the regular variables are shared across all of the configurations of this scenario, and the environmental variables are environment specific (e.g., specific to the use of the configuration, such as specific to staging versus production). Engine **1010** may expand the configuration and evaluate the constraints separately for each environment, where for different environmental inputs, engine **1010** outputs separate schemas for each environment. Moreover, for inputs that are shared by all environments (e.g., regular variables), engine **1010** may combine constraints across all of the environments into one constraint.

[0076] In some embodiments, engine **1010** may determine, based on the inputs, whether a given constraint is applicable by determining whether any candidate constraints are associated with a given input or set of inputs. Where there is a match, engine **1010** may output a set of constraints for the matching path. Engine **1010** may expand the relevant blueprint using the provided variables to obtain the configuration (e.g., by plugging the variables into relevant portions of the blueprint). For each guardrail, engine **1010** may

examine all instances that match the conditional part (e.g., associations between guardrails and conditions where they apply, as described in the foregoing), and may output a set of constraints for each match. A constraint may iterate over all instances of resources subject to guardrails (or sub-properties thereof).

[0077] As an example, take the guardrail 'when aws_s3_bucket.acl=="public", require name ends with "-public"'. This will examine all instances of aws_s3_bucket' in the configuration. For each with an 'acl' (Access Control List) property equal to public, the engine will output a constraint 'name endswith "-public"'. Engine **1010** may output these constraints in a variety of forms, including a granular form and a unified form. A granular form may have constraints from different guardrails be output separately, which allows the UI to provide info about the active guardrails to the user. A unified form may have constraints from all guardrails be combined and simplified, which allows the UI to present the user with the simplest "input widget" for inputting a valid value.

[0078] The guardrail schemas may be overlaid on top of blueprint **1002** to exclude values otherwise possible for a given resource within the context of blueprint **1002** that are not permitted by the applicable guardrails.

[0079] With the constraints defined, filtering module **908** filters a universe of possible values for each variable to exclude possible values that are outside of the corresponding constraints. In order to obtain the universe of possible values, filtering module **908** retrieves information from a knowledge graph (depicted as inputs **1003**) The knowledge graph may be a repository that stores allowable properties for different types of resources. For example, S3 buckets may have on the order of 70 configurable properties in scenarios where guardrails and policies are not a factor. The knowledge graph stores all of those configurable properties for each type of configurable resource.

[0080] Using the output of engine **1010**, filtering module **908** generates filtered values for each of the definable values by filtering values for each of the definable variables based on the mapping, thereby resulting in combined schemas **1021**, which take the guardrail schemas and overlay them with applicable information from knowledge graph **1003**. That is, for each variable that is definable for a resource to be created, knowledge graph **1003** may indicate all possible values. Knowledge graph **1003** may also indicate what variables are definable for a given resource, and which ones default to a given value. Filtering module **908** may thereby generate the combined schemas by limiting the values assignable to resource variables within the guardrail schemas to those that are permissible by the guardrails.

[0081] Definable variable **910** may, for each variable that can be defined for a blueprint, automatically fill in values where only one variable is permissible for the resource based on guardrails **1004**, and may associate viable values (e.g., two or more possible values) for user selection for variables where more than one variable is permissible.

[0082] Filtering module **908** may generate for display a user interface based on the combined schemas **1021**. The user interface may prompt a user to select values that comply with guardrails for each variable to defined for the resource. The prompts may prioritize a recommendation of a given value over other possible values using any mechanism discussed above in relation to a recommendation engine, such as activity of configuration recommendation module **210** and/or resource configuration module **810**.

[0083] In some embodiments, filtering module **908** may, one or more given variables, generating a filtered value (e.g., from the combined schema) by determining a plurality of candidate values that satisfy the constraints. Filtering module **908** may input the plurality of candidate values into a machine learning model, and receive as output a score for each of the plurality of candidate values. Filtering module **908** may pre-populate a candidate value having a highest score relative to other ones of the plurality of candidate values for the given variable, where a ranked list of the plurality of candidate values is displayed on the user interface responsive to selection of a user interface element for the given variable (e.g., a drop-down list that is ranked based on the scores). The machine learning model may be trained based on historical data of the user, or an entity (e.g., team) of which the user is a part, in selecting values for the given variable when making configurations for the same or semantically similar resources.

[0084] User interface module **910** may generate for display a user interface showing variable names alongside fields for defining the variables. The fields for defining the variables may be free-text field, a drop-down menu showing variables that fit within the combined schema, pre-filled values where only one possible value fits within the combined schema, or any other field that accepts definition of a variable. Using these fields, user interface module **910** may receive values for each variable (or each variable that requires definition, while pre-filling variables with default or only one value available or with recommendations tailored to the user).

[0085] In some embodiments, user interface module **910** may include an indication of constraints that apply to each definable variable. For example, user interface module **910** may reference the guardrail(s) for constraints for each variable, and may output for display an indication of those constraints in visual connection with each associated variable. Yet further, in some embodiments, user interface module **910** may additionally, or alternatively, output an indication of which guardrails are enforcing each of the constraints.

[0086] In some embodiments, user interface module **910** receives a plurality of a plurality of environmental variables (e.g., as part of variables **1001**), resulting in different schemas per environment. In such embodiments, different resulting combined schemas **1021** may require separate configuration. In such embodiments, user interface module **910** may generate for display different user interface for defining variable values for each environment to be configured. In such cases, where the different environments have common candidate values for a set of variables, a schema for shared inputs may drive a shared user interface for those variables. The shared user interface may be a separate user interface from the different environment user interfaces. In some embodiments, values from one environment that are selected may have their definitions carried to a user interface for other environments having variables with the same candidate values, where the definitions may be reconfigured at those other user interfaces on a per-environment basis.

[0087] Even having used engine **1010**, configurations may still violate one or more policies. Therefore, in some embodiments, validator module **914** takes one or more selected values of a configuration (and/or an entire configu-

ration) and inputs the selected values back into validated resources function **1022**. This enables the engine to validate the definitions against blueprint **1002**. In some embodiments, policy enforcement service **130** might fail to enforce a particular constraint. For example, there may be scenarios where there is no way of enforcing that constraint in the UX. For instance, imagine a constraint like "the bucket name may not contain the word 'silly'". Preventing a user from typing "silly" may not be possible, but policy enforcement service **130** may display an error when the word is detected in a bucket name. As another example of a constraint that cannot be enforced in the UX, consider a conditional constraint like "the bucket ACL may not be public if the bucket is attached to CDN". The user might have picked "public" before attaching the bucket to the CDN, when "public" was still valid. Policy enforcement service **130** would then show an error in the UX, informing the user that "public" is no longer valid and must be changed. At this point, the UX would restrict "public" from being selected. As another example, a conditional constraint may come into effect after a (now invalid) value has been entered. For example, "if a=true, b starts with 'foo' ". If the user has already entered 'bar' for b, and then sets a to true, the validator will detect the now invalid value for b.

[0088] The validated Resources block **1022** may validate each of the schemas (e.g., each guardrail, each knowledge graph constraint) against the configuration to output a collection of violations. A violation is a path to a property in the configuration and an associated constraint (e.g., guardrail) that was violated. A violation may be shown in the UX along with the path to enable the user to be directly linked to the source of the violation for ease in adjusting the underlying resource configuration. The UX may include an option for a user to override or disable constraints informing a violation (e.g., where a rule has an exception, and the guardrail is deemed unnecessary in a given situation). In some embodiments, user interface module **910** may navigate a user interface to configuration inputs resulting in the violation, and may output a recommendation for how to resolve the violation. In some embodiments, user interface module **910** may output a selectable option to override the violation. For example, where the user is credentialed to override the violation of a policy, an override option may be output.

Resource Parameter Recommendations

[0089] Returning to FIG. **8**, in some embodiments, resource configuration module **810** may generate an improved user interface having recommendations for various resource parameters. Some resource parameters may in turn leverage a dependent resource. For example, where a resource being configured is a virtual machine (VM), a parameter that is to be configured may include a Virtual Private Cloud (VPC) to which the VM is to be deployed. Some VPCs may themselves have attributes that violate a guardrail of the VM's blueprint, and therefore are unusable. The recommendation engine factors in dependent resources from different possible configuration parameters in order to provide recommendations that are compliant with a blueprint for the resource.

[0090] FIG. **12** illustrates one embodiment of sub-modules of a resource configuration module, in accordance with some embodiments of the disclosure. As depicted in FIG. **12**, resource configuration module **810** may include parameter selection module **1202**, configuration option module **1204**, dependent resource determination module **1206**, blueprint harmonization module **1208**, and configuration recommendation module **1210**. Fewer or more modules and/or databases may be used to achieve the disclosed operation of resource configuration module **810**.

[0091] Parameter selection module **1202** may generate for display a user interface having configuration fields for configuring parameters of a resource. In some scenarios, the resource has a blueprint defining guardrails; however, wherever such a blueprint is mentioned, a set of guardrails may have been defined even where no blueprint exists. An exemplary user interface is shown in FIG. **13**. FIG. **13** illustrates an exemplary user interface for configuring a resource parameter associated with a downstream resource. As illustrated in FIG. **13**, user interface **1300** is a user interface for configuring subnet parameters of an instance. This is merely for convenience, and user interface **1300** in practice may be a user interface for configuring any parameters of any resource (e.g., parameters of a bucket to be generated, of a virtual machine to be created, and so on). At least some of the resource parameters may be pre-populated based on a blueprint (or based a set of guardrails that did not originate from a blueprint) for the resource, as described in the foregoing. For example, where only one compliant value for a parameter is in compliance with guardrails, that parameter may be pre-populated. As another example, where a recommended value is determined and is in compliance with the guardrails (e.g., of the blueprint or standalone guardrails), the recommended value may be pre-populated. In some embodiments, user interface **1300** may be a part of user interface **700**.

[0092] Parameter selection module **1202** may receive an input with respect to a configuration field for a parameter. The input may be any interaction with a field, such as selecting a dropdown component, interacting with a field in a manner that enables textual input, and/or selection of any other option or manner for defining the parameter. Either prior to or in response to receiving the input, configuration option module **1204** may determine a set of configuration options for the parameter. The set of configuration parameters may be determined at least in part from a knowledge graph (e.g., knowledge graph **1003**). Determining configuration options for the parameter may be a resource-intensive operation, and therefore activities of configuration option module **1204** may occur responsive to receiving the input. As an example, a default value or recommended value may be input without performing the resource-intense configuration option for a field, and therefore processing for configuration option determination may be limited to those fields that are selected, thereby significantly saving on computational expense.

[0093] In some embodiments, prior to referencing the knowledge graph, configuration option module **1204** may scan a repository associated with the user (e.g., the user's repository, a team's or enterprise's repository that is associated with the user, etc.) for existing resources. Configuration option module **1204** may additionally or alternatively scan the blueprint (or other set of guardrails) for the resource currently being configured for other resources that are currently being configured. Configuration option module **1204** may, based on the existing resources and the other resources being configured, reference the knowledge graph to determine which existing resources can be referenced by a given input for the parameter that is being configured.

[0094] The knowledge graph may indicate all possibilities of properties for this parameter of the resource, those possibilities making up the configuration options. For example, all known possible subnet names may be determined. The resource may have a blueprint that indicates guardrails, or guardrails may be defined without a blueprint or in addition to a blueprint. For subnet names for example, the guardrails may require that the resource be on a private subnet, and therefore the set of resources may be reduced by configuration option module **1204** to indicate only private subnets, thereby resulting in configuration possibilities formed using the private subnets. In some embodiments, the knowledge graph may include edges between parameters and their corresponding fields. Resource configuration module **810** may leverage these edges to determine a set of configuration options for other parameters using these edges. In some embodiments, configuration option module **1204** may reference the knowledge graph using a resource type as a filter, so as to compare for nodes within the knowledge graph having a parameter type consistent with the resource type and not for other nodes, thereby reducing the complexity of the reference operation.

[0095] In some embodiments, the knowledge graph may indicate a downstream resource. For example, when configuring the VPC ID parameter of user interface **1300**, dependent resource determination module **1206** may determine that a VPC itself might need to be configured to host the instance being created, the VPC being a downstream dependent resource. The VPC will in turn need to have features that are compliant with guardrails of the blueprint for the resource being configured. Therefore, for each option of the set of configuration options determined using the knowledge graph, dependent resource determination module **1206** determines what dependent resources are implicated, and then determines which of those dependent resources are compliant with the guardrails for the resource in order for configuration option module **1204** to determine a subset of the options that are viable for resource configuration.

[0096] In some embodiments, for each dependent resource, configuration option module **1204** retrieves a blueprint (or other set of guardrails) for the dependent resource and determines whether the blueprint (or other set of guardrails) violates the guardrails for the resource being configured. Following the VPC example, blueprints for VPC resources that could be instantiated to host an instance being configured using user interface **1300** may be retrieved. Blueprint harmonization module **1208** may determine a subset of the set of configuration options that are viable for configuring the parameter, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrail. Blueprint harmonization module **1208** may determine non-compliant resources by determining that the resources' given blueprint does not allow for a value of at least one parameter in compliance with the blueprint of the resource. In some embodiments, rather than retrieve blueprints where dependent resources are already configured, blueprint harmonization module **1208** may retrieve existing configurations for those dependent resources and determine whether those existing configurations violate the guardrails of the resource currently being configured. Blueprint harmonization module **1208** may operate equally without blueprints, working to instead harmonize guardrails that are not formed from blueprints.

[0097] In some embodiments, configuration option module **1204** may exclude a dependent resource as a configuration option despite having a blueprint (or other set of guardrails) that is consistent with the guardrails for the primary resource. Configuration option module **1204** may detect that a dependent resource, or the resource's blueprint (or other set of guardrails), connects directly or indirectly to the primary resource, and would thus create a link cycle. Link cycles are invalid, and would be rejected by a run tool (e.g., Terraform), and a Cloud Provider API request to effect changes would fail. Even if this were not the case, link cycles would be detrimental because they may cause the primary resource to become overloaded and crash, and other operational inefficiencies may occur. A direct link may occur where the primary resource is A, the dependent resource is B, and B is configured to have A as a downstream dependency. An indirect link may occur where the primary resource is A, the dependent resource is B, and a tertiary resource is C, where A links to C, and C links to B; in such a scenario, if A links to B, and one of B's properties could link to A, a link cycle could occur and therefore dependent resource B should be excluded as a configuration option despite otherwise having guardrail compliance.

[0098] Configuration recommendation module **1210** may generate for display using the user interface a recommendation for the configuration field. In some embodiments, the recommendation may include the entire subset of configuration options that are viable following blueprint harmonization. In some embodiments, the recommendation may be a ranked list of recommendations. The ranking may occur using machine learning and/or heuristics based on a profile of the user and/or activity of other users within a same team or environment of the user, as described with respect to other recommendations made in this disclosure. For example, features of a profile of the user or team such as historical selection may act as training examples to a machine learning model, the training examples labeled with whether or not the user selected a given configuration option for a parameter. Configuration recommendation module **1210** may output a recommendation, such as a ranked list or single best recommendation, based on probabilities output by the machine learning model corresponding to each configuration option of the subset.

[0099] Determining dependent resources may be a resource-intensive operation, particularly where there are many downstream resources for many configuration options for a given parameter. In some embodiments, as dependent resources are determined, dependent resource determination module **1206** may store links between a parameter configuration option and a dependent resource as a training example in a training repository. Resource configuration module **810** may train a machine learning model, using the training examples, to accept an input of guardrails and dependent resources as input, and to output the set of downstream resources that is in compliance with the guardrails of the primary resource. This may improve efficiency in readying resource configuration module **810** to move to the next step of deriving the subset from the downstream resources depending on their compliance with the resource's guardrails.

[0100] FIG. **14** illustrates a flowchart showing an exemplary process for generating parameter configuration recommendations based on dependent resource conflicts with a resource blueprint, in accordance with an embodiment.

Process **1400** may be performed by one or more processors executing instructions stored on a non-transitory computer-readable medium. The instructions, when executed, may cause the modules and sub-modules of policy enforcement service **130** to perform their respective operations. Process **1400** begins with policy enforcement service **130** generating for display **1410** a user interface having configuration fields for configuring parameters of a resource (e.g., using parameter selection module **1202**), the resource having a blueprint defining guardrails. Policy enforcement service **130** determines **1420** a set of configuration options for a parameter of the parameters of the resource (e.g., using configuration option module **1204**.

[0101] Policy enforcement service **130** determines **1430**, for each option of the set of configuration options, a dependent resource (e.g., using dependent resource determination module **1206**. For each dependent resource, policy enforcement service **130** retrieves **1440** a blueprint (or other set of guardrails) for the dependent resource and determines whether the blueprint (or other set of guardrails) for the dependent resource violates the guardrails (e.g., using blueprint harmonization module **1208**). Policy enforcement service **130** determines **1450** a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails (e.g., using blueprint harmonization module **1208**). Policy enforcement service **130** generates for display **1460** using the user interface a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset of the set of configuration options (e.g., using configuration recommendation module **1210**).

COMPUTING MACHINE ARCHITECTURE

[0102] FIG. **3** is a block diagram illustrating components of an example machine able to read instructions from a machine-readable medium and execute them in a processor (or controller). Specifically, FIG. **3** shows a diagrammatic representation of a machine in the example form of a computer system **300** within which program code (e.g., software) for causing the machine to perform any one or more of the methodologies discussed herein may be executed. The program code may be comprised of instructions **324** executable by one or more processors **302**. In alternative embodiments, the machine operates as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the machine may operate in the capacity of a server machine or a client machine in a server-client network environment, or as a peer machine in a peer-to-peer (or distributed) network environment.

[0103] The machine may be a server computer, a client computer, a personal computer (PC), a tablet PC, a set-top box (STB), a personal digital assistant (PDA), a cellular telephone, a smartphone, a web appliance, a network router, switch or bridge, or any machine capable of executing instructions **324** (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated, the term "machine" shall also be taken to include any collection of machines that individually or jointly execute instructions **124** to perform any one or more of the methodologies discussed herein.

[0104] The example computer system **300** includes a processor **302** (e.g., a central processing unit (CPU), a graphics processing unit (GPU), a digital signal processor (DSP), one or more application specific integrated circuits (ASICs), one or more radio-frequency integrated circuits (RFICs), or any combination of these), a main memory **304**, and a static memory **306**, which are configured to communicate with each other via a bus **308**. The computer system **300** may further include visual display interface **310**. The visual interface may include a software driver that enables displaying user interfaces on a screen (or display). The visual interface may display user interfaces directly (e.g., on the screen) or indirectly on a surface, window, or the like (e.g., via a visual projection unit). For ease of discussion the visual interface may be described as a screen. The visual interface **310** may include or may interface with a touch enabled screen. The computer system **300** may also include alphanumeric input device **312** (e.g., a keyboard or touch screen keyboard), a cursor control device **314** (e.g., a mouse, a trackball, a joystick, a motion sensor, or other pointing instrument), a storage unit **316**, a signal generation device **318** (e.g., a speaker), and a network interface device **320**, which also are configured to communicate via the bus **308**.

[0105] The storage unit **316** includes a machine-readable medium **322** on which is stored instructions **324** (e.g., software) embodying any one or more of the methodologies or functions described herein. The instructions **324** (e.g., software) may also reside, completely or at least partially, within the main memory **304** or within the processor **302** (e.g., within a processor's cache memory) during execution thereof by the computer system **300**, the main memory **304** and the processor **302** also constituting machine-readable media. The instructions **324** (e.g., software) may be transmitted or received over a network **326** via the network interface device **320**.

[0106] While machine-readable medium **322** is shown in an example embodiment to be a single medium, the term "machine-readable medium" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, or associated caches and servers) able to store instructions (e.g., instructions **324**). The term "machine-readable medium" shall also be taken to include any medium that is capable of storing instructions (e.g., instructions **324**) for execution by the machine and that cause the machine to perform any one or more of the methodologies disclosed herein. The term "machine-readable medium" includes, but not be limited to, data repositories in the form of solid-state memories, optical media, and magnetic media.

ADDITIONAL CONFIGURATION CONSIDERATIONS

[0107] Throughout this specification, plural instances may implement components, operations, or structures described as a single instance. Although individual operations of one or more methods are illustrated and described as separate operations, one or more of the individual operations may be performed concurrently, and nothing requires that the operations be performed in the order illustrated. Structures and functionality presented as separate components in example configurations may be implemented as a combined structure or component. Similarly, structures and functionality presented as a single component may be implemented as separate components. These and other variations, modifications, additions, and improvements fall within the scope of the subject matter herein.

[0108] Certain embodiments are described herein as including logic or a number of components, modules, or mechanisms. Modules may constitute either software modules (e.g., code embodied on a machine-readable medium or in a transmission signal) or hardware modules. A hardware module is tangible unit capable of performing certain operations and may be configured or arranged in a certain manner. In example embodiments, one or more computer systems (e.g., a standalone, client or server computer system) or one or more hardware modules of a computer system (e.g., a processor or a group of processors) may be configured by software (e.g., an application or application portion) as a hardware module that operates to perform certain operations as described herein.

[0109] In various embodiments, a hardware module may be implemented mechanically or electronically. For example, a hardware module may comprise dedicated circuitry or logic that is permanently configured (e.g., as a special-purpose processor, such as a field programmable gate array (FPGA) or an application-specific integrated circuit (ASIC)) to perform certain operations. A hardware module may also comprise programmable logic or circuitry (e.g., as encompassed within a general-purpose processor or other programmable processor) that is temporarily configured by software to perform certain operations. It will be appreciated that the decision to implement a hardware module mechanically, in dedicated and permanently configured circuitry, or in temporarily configured circuitry (e.g., configured by software) may be driven by cost and time considerations.

[0110] Accordingly, the term "hardware module" should be understood to encompass a tangible entity, be that an entity that is physically constructed, permanently configured (e.g., hardwired), or temporarily configured (e.g., programmed) to operate in a certain manner or to perform certain operations described herein. As used herein, "hardware-implemented module" refers to a hardware module. Considering embodiments in which hardware modules are temporarily configured (e.g., programmed), each of the hardware modules need not be configured or instantiated at any one instance in time. For example, where the hardware modules comprise a general-purpose processor configured using software, the general-purpose processor may be configured as respective different hardware modules at different times. Software may accordingly configure a processor, for example, to constitute a particular hardware module at one instance of time and to constitute a different hardware module at a different instance of time.

[0111] Hardware modules can provide information to, and receive information from, other hardware modules. Accordingly, the described hardware modules may be regarded as being communicatively coupled. Where multiple of such hardware modules exist contemporaneously, communications may be achieved through signal transmission (e.g., over appropriate circuits and buses) that connect the hardware modules. In embodiments in which multiple hardware modules are configured or instantiated at different times, communications between such hardware modules may be achieved, for example, through the storage and retrieval of information in memory structures to which the multiple hardware modules have access. For example, one hardware module may perform an operation and store the output of that operation in a memory device to which it is communicatively coupled. A further hardware module may then, at a later time, access the memory device to retrieve and process the stored output. Hardware modules may also initiate communications with input or output devices, and can operate on a resource (e.g., a collection of information).

[0112] The various operations of example methods described herein may be performed, at least partially, by one or more processors that are temporarily configured (e.g., by software) or permanently configured to perform the relevant operations. Whether temporarily or permanently configured, such processors may constitute processor-implemented modules that operate to perform one or more operations or functions. The modules referred to herein may, in some example embodiments, comprise processor-implemented modules.

[0113] Similarly, the methods described herein may be at least partially processor- implemented. For example, at least some of the operations of a method may be performed by one or processors or processor-implemented hardware modules. The performance of certain of the operations may be distributed among the one or more processors, not only residing within a single machine, but deployed across a number of machines. In some example embodiments, the processor or processors may be located in a single location (e.g., within a home environment, an office environment or as a server farm), while in other embodiments the processors may be distributed across a number of locations.

[0114] The one or more processors may also operate to support performance of the relevant operations in a "cloud computing" environment or as a "software as a service" (SaaS). For example, at least some of the operations may be performed by a group of computers (as examples of machines including processors), these operations being accessible via a network (e.g., the Internet) and via one or more appropriate interfaces (e.g., application program interfaces (APIs).)

[0115] The performance of certain of the operations may be distributed among the one or more processors, not only residing within a single machine, but deployed across a number of machines. In some example embodiments, the one or more processors or processor-implemented modules may be located in a single geographic location (e.g., within a home environment, an office environment, or a server farm). In other example embodiments, the one or more processors or processor-implemented modules may be distributed across a number of geographic locations.

[0116] Some portions of this specification are presented in terms of algorithms or symbolic representations of operations on data stored as bits or binary digital signals within a machine memory (e.g., a computer memory). These algorithms or symbolic representations are examples of techniques used by those of ordinary skill in the data processing arts to convey the substance of their work to others skilled in the art. As used herein, an "algorithm" is a self-consistent sequence of operations or similar processing leading to a desired result. In this context, algorithms and operations involve physical manipulation of physical quantities. Typically, but not necessarily, such quantities may take the form of electrical, magnetic, or optical signals capable of being stored, accessed, transferred, combined, compared, or otherwise manipulated by a machine. It is convenient at times, principally for reasons of common usage, to refer to such signals using words such as "data," "content," "bits," "values," "elements," "symbols," "characters," "terms," "numbers," "numerals," or the like. These words, however, are

merely convenient labels and are to be associated with appropriate physical quantities.

[0117] Unless specifically stated otherwise, discussions herein using words such as "processing," "computing," "calculating," "determining," "presenting," "displaying," or the like may refer to actions or processes of a machine (e.g., a computer) that manipulates or transforms data represented as physical (e.g., electronic, magnetic, or optical) quantities within one or more memories (e.g., volatile memory, non-volatile memory, or a combination thereof), registers, or other machine components that receive, store, transmit, or display information.

[0118] As used herein any reference to "one embodiment" or "an embodiment" means that a particular element, feature, structure, or characteristic described in connection with the embodiment is included in at least one embodiment. The appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same embodiment.

[0119] Some embodiments may be described using the expression "coupled" and "connected" along with their derivatives. It should be understood that these terms are not intended as synonyms for each other. For example, some embodiments may be described using the term "connected" to indicate that two or more elements are in direct physical or electrical contact with each other. In another example, some embodiments may be described using the term "coupled" to indicate that two or more elements are in direct physical or electrical contact. The term "coupled," however, may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other. The embodiments are not limited in this context.

[0120] As used herein, the terms "comprises," "comprising," "includes," "including," "has," "having" or any other variation thereof, are intended to cover a non-exclusive inclusion. For example, a process, method, article, or apparatus that comprises a list of elements is not necessarily limited to only those elements but may include other elements not expressly listed or inherent to such process, method, article, or apparatus. Further, unless expressly stated to the contrary, "or" refers to an inclusive or and not to an exclusive or. For example, a condition A or B is satisfied by any one of the following: A is true (or present) and B is false (or not present), A is false (or not present) and B is true (or present), and both A and B are true (or present).

[0121] In addition, use of the "a" or "an" are employed to describe elements and components of the embodiments herein. This is done merely for convenience and to give a general sense of the invention. This description should be read to include one or at least one and the singular also includes the plural unless it is obvious that it is meant otherwise.

[0122] Upon reading this disclosure, those of skill in the art will appreciate still additional alternative structural and functional designs for a system and a process for reconciling configuration settings for imported resources through the disclosed principles herein. Thus, while particular embodiments and applications have been illustrated and described, it is to be understood that the disclosed embodiments are not limited to the precise construction and components disclosed herein. Various modifications, changes and variations, which will be apparent to those skilled in the art, may be made in the arrangement, operation and details of the method and apparatus disclosed herein without departing from the spirit and scope defined in the appended claims.

What is claimed is:

1. A method comprising:

generating for display a user interface having configuration fields for configuring parameters of a resource, the resource having a blueprint defining guardrails;

determining a set of configuration options for a parameter of the parameters of the resource;

determining, for each option of the set of configuration options, a dependent resource;

for each dependent resource, retrieving a blueprint for the dependent resource and determining whether the blueprint for the dependent resource violates the guardrails;

determining a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails; and

generating for display using the user interface a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset of the set of configuration options.

2. The method of claim **1**, wherein the configuration fields are at least partially pre-populated based on a blueprint for the resource.

3. The method of claim **1**, wherein determining the set of configuration options for the parameter of the parameters of the resource occurs responsive to receiving a selection of an option within the user interface to define the parameter.

4. The method of claim **1**, wherein determining the set of configuration options comprises comparing a resource type for the parameter to parameter types in a knowledge graph.

5. The method of claim **1**, wherein determining the set of configuration options comprises inputting the parameter into a supervised machine learning model and receiving as output from the model the set of configuration options.

6. The method of claim **1**, wherein determining, for a given dependent resource, that its given blueprint violates the guardrail comprises determining that the given blueprint does not allow for a value of at least one parameter in compliance with the blueprint of the resource.

7. The method of claim **1**, wherein the subset of the set of configuration options further excludes other options of the set of configuration options that are associated with dependent resources that could create a link cycle.

8. A non-transitory computer-readable medium comprising memory with instructions encoded thereon that, when executed by one or more processors, cause the one or more processors to perform operations, the instructions comprising instructions to:

generate for display a user interface having configuration fields for configuring parameters of a resource, the resource having a blueprint defining guardrails;

determine a set of configuration options for a parameter of the parameters of the resource;

determine, for each option of the set of configuration options, a dependent resource;

for each dependent resource, retrieve a blueprint for the dependent resource and determining whether the blueprint for the dependent resource violates the guardrails;

determine a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails; and

generate for display using the user interface a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset of the set of configuration options.

9. The non-transitory computer-readable medium of claim **8**, wherein the configuration fields are at least partially pre-populated based on a blueprint for the resource.

10. The non-transitory computer-readable medium of claim **8**, wherein determining the set of configuration options for the parameter of the parameters of the resource occurs responsive to receiving a selection of an option within the user interface to define the parameter.

11. The non-transitory computer-readable medium of claim **8**, wherein the instructions to determine the set of configuration options comprise instructions to compare a resource type for the parameter to parameter types in a knowledge graph.

12. The non-transitory computer-readable medium of claim **8**, wherein the instructions to determine the set of configuration options comprise instructions to input the parameter into a supervised machine learning model and receiving as output from the model the set of configuration options.

13. The non-transitory computer-readable medium of claim **8**, wherein the instructions to determine, for a given dependent resource, that its given blueprint violates the guardrail comprise instructions to determine that the given blueprint does not allow for a value of at least one parameter in compliance with the blueprint of the resource.

14. The non-transitory computer-readable medium of claim **8**, wherein the subset of the set of configuration options further excludes other options of the set of configuration options that are associated with dependent resources that could create a link cycle.

15. A system comprising:

memory with instructions encoded thereon; and

one or more processors that, when executing the instructions, are caused to perform operations comprising:

generating for display a user interface having configuration fields for configuring parameters of a resource, the resource having a blueprint defining guardrails;

determining a set of configuration options for a parameter of the parameters of the resource;

determining, for each option of the set of configuration options, a dependent resource;

for each dependent resource, retrieving a blueprint for the dependent resource and determining whether the blueprint for the dependent resource violates the guardrails;

determining a subset of the set of configuration options, the subset excluding options of the set of configuration options associated with dependent resources that violate the guardrails; and

generating for display using the user interface a recommendation for a configuration field of the configuration fields corresponding to the parameter based on the subset of the set of configuration options.

16. The system of claim **15**, wherein the configuration fields are at least partially pre-populated based on a blueprint for the resource.

17. The system of claim **15**, wherein determining the set of configuration options for the parameter of the parameters of the resource occurs responsive to receiving a selection of an option within the user interface to define the parameter.

18. The system of claim **15**, wherein determining the set of configuration options comprises comparing a resource type for the parameter to parameter types in a knowledge graph.

19. The system of claim **15**, wherein determining the set of configuration options comprises inputting the parameter into a supervised machine learning model and receiving as output from the model the set of configuration options.

20. The system of claim **15**. wherein determining. for a given dependent resource, that its given blueprint violates the guardrail comprises determining that the given blueprint does not allow for a value of at least one parameter in compliance with the blueprint of the resource.

* * * * *