(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
27 February 2020 (27.02.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/041073 A1

(51) International Patent Classification:
G06F 9/54 (2006.01) *H04L 29/06* (2006.01)

(21) International Application Number:
PCT/US2019/046565

(22) International Filing Date:
14 August 2019 (14.08.2019)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
16/112,408 24 August 2018 (24.08.2018) US
16/112,396 24 August 2018 (24.08.2018) US

(71) Applicant: **VMWARE, INC.** [US/US]; 3401 Hillview Avenue, Palo Alto, CA 94304 (US).

(72) Inventors: **MYNENI, Sirisha**; 3401 Hillview Avenue, Palo Alto, CA 94304 (US). **CHANDA, Arijit**; 3274 Lago Deomo Place, San Jose, CA 95136 (US). **GUNDA, Laxmikant, Vithal**; 3401 Hillview Avenue, Palo Alto, CA 94304 (US). **POON, Arnold**; 3401 Hillview Avenue, Palo Alto, CA 94304 (US). **GHANNADIAN, Farzad**; 3401 Hillview Avenue, Palo Alto, CA 94304 (US). **KUMAR, Kausum**; 3401 HILLVIEW avenue, apal alto, CA 94304 (US).

(74) Agent: **ADELI, Mani** et al.; Adeli Llp, 11859 Wilshire Blvd., Suite 408, Los Angeles, CA 90025 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP,

(54) Title: HIERARCHICAL API FOR DEFINING A MULTI-SEGMENTED APPLICATION IN AN SDDC

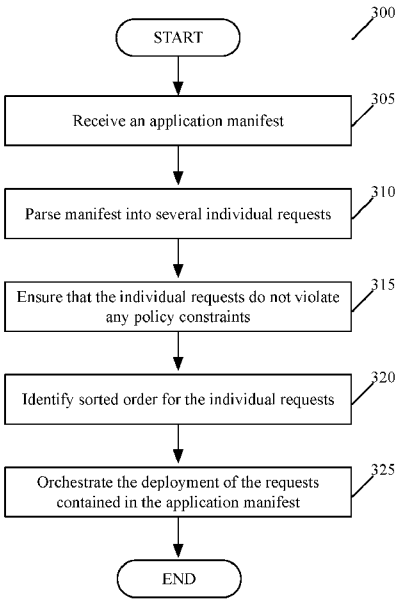


Figure 3

(57) **Abstract:** Some embodiments provide a simplified mechanism to deploy and control a multi- segmented application by using application-based manifests that express how application segments of the multi-segment application are to be defined or modified, and how the communication profiles between these segments. In some embodiments, these manifests are application specific. Also, in some embodiments, deployment managers in a software defined datacenter (SDDC) provide these manifests as templates to administrators, who can use these templates to express their intent when they are deploying multi-segment applications in the datacenter. Application-based manifests can also be used to control previously deployed multi- segmented applications in the SDDC. Using such manifests would enable the administrators to be able to manage fine grained micro-segmentation rules based on endpoint and network attributes.

WO 2020/041073 A1

[Continued on next page]

KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

HIERARCHICAL API FOR DEFINING A MULTI-SEGMENTED APPLICATION IN AN SDDC

BACKGROUND

[0001] In enterprise data centers, firewalls have been the essence for the networking and security of the applications that run on them. It started with Access Control Lists (ACLs), which provided rules that are applied to port numbers or IP addresses that are available on a host or other layer 3, each with a list of hosts and/or networks permitted to use the service. After ACLs, macro-segmentation came along to provide IP based enforcement for every application that runs on a host. This enabled granular level controls for enterprise administrators to protect their workloads based on VLANs.

[0002] With network virtualization, micro-segmentation has turned around the networking and security space by providing the ability to enforce distributed firewall rules across hosts in the datacenter based on L4-L7 network services and attributes. There are new firewalls which have the capability to perform deep packet introspection in the transport layer and include Web application filtering, Verb based firewalls and URL filtering.

[0003] **Figure 1** shows the current workflow for specifying firewall controls for micro-segmented applications. As shown, the administrator has to first define (at 105) an intent as to which application they would like to secure. Based on the intent, the administrator has to create (at 110) domains and groups to define the boundaries of each component of the application. Once the groups are created, the administrator then defines (at 115) how these components can communicate with each other based on a communication profile.

[0004] After the profile and groups are created, the resulting policy is published at 120 to the software defined datacenter (SDDC) network managers 150. After publishing the policy, the administrator has to then login to the network manager for every instance of the datacenter the manager controls and create (at 125) the networking and security groups based on the grouping criteria specified in the creation of the domains and groups. The criteria could be based on logical switch ports, tags, or VM/container names.

[0005] The administrator then has to manually manage (130) the workload VMs/containers by applying the corresponding tags so that they match the criteria during the creation of the network and security groups at 125. When the tag matches the criteria, the firewall rule defined in the communication profile is then applied (at 135) to respective VMs.

[0006] This approach has several shortcomings. For instance, the management of grouping criteria (Tags, VM name etc.) is manual and cumbersome. This is particularly problematic as this management has to be repeated across multiple environments(e.g., development, staging and production). In addition, discovery and classification of the applications is an administrative overhead and is often error prone. This approach is also not scalable for dynamic workload (e.g. Container) environments when the entities being protected are ephemeral in nature

BRIEF SUMMARY

[0008] Some embodiments provide a simplified mechanism to deploy and control a multi-segmented application by using application-based manifests that express how application segments of the multi-segment application are to be defined or modified, and how the communication profiles between these segments. In some embodiments, these manifests are application specific. Also, in some embodiments, deployment managers in a software defined datacenter (SDDC) provide these manifests as templates to administrators, who can use these templates to express their intent when they are deploying multi-segment applications in the datacenter. Application-based manifests can also be used to control previously deployed multi-segmented applications in the SDDC. Using such manifests would enable the administrators to be able to manage fine grained micro-segmentation rules based on endpoint and network attributes.

[0009] Multi-segment application is an application that includes multiple application segments. In some embodiments, each of one or more application segments is a standalone application executing in its own memory space that is disjoint from the memory spaces of any other application segment of the multi-segment application. In some embodiments, different application segments of a multi-segment application are implemented by different machines (e.g., different VMs or containers).

[0010] In some embodiments, an application manifest includes a syntactical representation of the multi-segmented application, which may be defined after implementing the manifest, or it may have been defined earlier. An application manifest in some embodiments is a hierarchical API that includes two or more commands that define or modify (1) one or more application segments, and (2) one or more policies associated with them. The application manifest is a hierarchical API as different commands can be nested under other commands, e.g., the definition of one group of applications can include the definition of specific machines (e.g., specific VMs) to implement specific applications in the group. In some embodiments, the application manifests are provided to administrators as predefined templates that encapsulate well-known applications and their dependencies, as well as an ability to model the applications based on the administrator's requirements.

[0011] In some embodiments, the manifest is defined in a declarative language. In some embodiments, a manifest-processing framework in the SDDC parses the manifest into several commands that (1) direct compute managers in the SDDC to deploy and configure the application

segments of the multi-segment application defined in the manifest, and (2) direct the network managers in the SDDC to define and deploy the network forwarding and service rules for implementing the communication profiles between the application segments as specified by the manifest and between application segments and other applications.

[0012] In some embodiments, the application segments are deployed as VMs or containers executing on host computers, and/or as standalone computers, in the SDDC. Similarly, the network forwarding and service rules in some embodiments are processed by software forwarding elements (e.g., software switches and routers) and software middlebox service VMs (e.g., service containers and/or services modules executing on the host computers in the SDDC). These forwarding and/or service rules are also configured in some embodiments on hardware forwarding elements (e.g., top-of-rack switches), standalone hardware or software gateways, and/or standalone middlebox appliances in the SDDC.

[0013] Some embodiments of the invention provide a method for deploying multi-segmented application in an SDDC. The method initially receives a hierarchical API command that, in a declarative format, specifies several operation requests to define several application segments of the multi-segmented application. The method parses the API command to identify the application segments. Based on the parsed API command, the method deploys several software defined (SD) resources that are needed to deploy several application segments, as well as forwarding and servicing operations between these segments.

[0014] The deployment process that the method uses in some embodiments that ensures that any first SD resource on which a second SD resource depends is deployed before the second resource. In some embodiments, a second SD resource depends on a first SD resource when the second SD resource is a child of the first SD resource. Alternatively, or conjunctively, a second SD resource can also depend on a first SD resource in some embodiments when the second SD resource has some operational dependency on the first SD resource.

[0015] In some embodiments, the method parses the API command by identifying several sets of SD resources, with each set having one or more SD resources at one resource level. The deployment in some embodiments deploys the identified SD resource sets at higher resource levels before deploying SD resources at lower resource levels. Examples of SD resources that can be specified in the hierarchical API command include SD compute elements (e.g., VMs or containers) to implement the application segments, SD forwarding elements (e.g., managed software switches

and routers, logical switches and routers implemented by the managed software switches and routers, etc.) to implement the forwarding rules for forwarding data messages associated with the application segments, and SD service middlebox modules (e.g., service VMs or modules that perform middlebox service operations such as firewall operations, load balancing operations, network address translation operations, encryption operations, intrusion detection operations, intrusion prevention operations, etc.) to enforce service rules for performing services on data messages associated with the application segments.

[0016] In some embodiments, an API processing system processes the API command. This command can include a set of parameters to update an earlier deployed SD resource. When this is the case, the API processing system deploys a multi-segmented application by using a set of parameters specified in the parsed API command to update the SD resource deployed earlier for the multi-segmented application. In some such cases, the API command includes a set of parameters that define a new SD resource. In such cases, the API processing system deploys the SD resource by deploying the SD resource based on the set of parameters specified in the parsed API command.

[0017] In some embodiments, the hierarchical API command is processed as one atomic unit. Accordingly, the API processing system determines whether the identified SD resources in the hierarchical API command are deployable. If so, the API processing system sends a confirmation that the API command has been successfully processed to a source that produced the hierarchical API command. On the other hand, when one or more SD resources in the API command are not deployable, the API processing system sends a message that the API command has not been successfully processed to the source that produced the hierarchical API command.

[0018] Some embodiments pave the way for the next generation of micro-segmentation by tying the context from the data compute endpoint to the network. The endpoint based context could be related to user identity, as well as application specific attributes such as file hash, publisher info, licensing, and process information. In some embodiments, the contextual information is used by application-based firewalls that are deployed in a distributer manned in the SDDC (e.g., through a virtualized network apparatus). One of the biggest challenges of commoditizing an application-based firewall is the consumption model for such a complex firewall, because there can be thousands of processes that might be running inside an endpoint and millions in a datacenter. However, by using application manifests, some embodiments provide a novel approach to allow

administrators to manage the very difficult task of creating fine grained context-based rules and managing these rules.

[0019] The preceding Summary is intended to serve as a brief introduction to some embodiments of the invention. It is not meant to be an introduction or overview of all inventive subject matter disclosed in this document. The Detailed Description that follows and the Drawings that are referred to in the Detailed Description will further describe the embodiments described in the Summary as well as other embodiments. Accordingly, to understand all the embodiments described by this document, a full review of the Summary, the Detailed Description, the Drawings, and the Claims is needed. Moreover, the claimed subject matters are not to be limited by the illustrative details in the Summary, the Detailed Description, and the Drawings.

BRIEF DESCRIPTION OF FIGURES

[0020] The novel features of the invention are set forth in the appended claims. However, for purposes of explanation, several embodiments of the invention are set forth in the following figures.

[0021] **Figure 1** shows the current workflow for specifying firewall controls for micro-segmented applications.

[0022] **Figure 2** illustrates a manifest-processing framework that processes application manifests received from tenant administrators in an SDDC.

[0023] **Figure 3** presents a process that illustrates the operation of the components of the manifest-processing framework.

[0024] **Figure 4** illustrates an example of an application manifest that defines a multi-segmented application called Slack.

[0025] **Figure 5** illustrates an example of Slack.

[0026] **Figure 6** illustrates the data model of the application illustrated in **Figure 5**.

[0027] **Figure 7** illustrates a process that represents an exemplary flow for defining and deploying a multi-segmented application based on a manifest template.

[0028] **Figure 8** illustrates the architecture of this new classification engine for some embodiments.

[0029] **Figure 9** illustrates an example of the API processing system of some embodiments of the invention.

[0030] **Figure 10** illustrates how some embodiments enforce context-based firewall rules on host computers.

[0031] **Figure 11** illustrates an example of data collection for learning about deployments of multi-segmented applications in a datacenter.

[0032] **Figure 12** conceptually illustrates a computer system with which some embodiments of the invention are implemented.

DETAILED DESCRIPTION

[0033] In the following detailed description of the invention, numerous details, examples, and embodiments of the invention are set forth and described. However, it will be clear and apparent to one skilled in the art that the invention is not limited to the embodiments set forth and that the invention may be practiced without some of the specific details and examples discussed.

[0034] Some embodiments provide a novel application-based manifest, which provides a simplified mechanism to deploy and control a multi-segment application and to define the communication profiles between the segments of the multi-segment application. A multi-segment application is an application that includes multiple application segments. In some embodiments, each application segment can be a standalone application executing in its own memory space that is disjoint from the memory spaces of any other application segment of the multi-segment application. In some embodiments, different application segments of a multi-segment application are implemented by different machines (e.g., different VMs or containers).

[0035] In some embodiments, deployment managers in a software defined datacenter (SDDC) provide these manifests as templates to administrators. The administrators, in turn, can use these templates to express their intent when they are deploying multi-segment applications in the datacenter. Application-based manifests can also be used to control previously deployed multi-segmented applications in the SDDC. Using such manifests would enable the administrators to be able to manage fine grained micro-segmentation rules based on endpoint and network attributes.

[0036] In this document, data messages refer to a collection of bits in a particular format sent across a network. One of ordinary skill in the art will recognize that the term data message may be used herein to refer to various formatted collections of bits that may be sent across a network, such as Ethernet frames, IP packets, TCP segments, UDP datagrams, etc. Also, as used in this document, references to L2, L3, L4, and L7 layers (or layer 2, layer 3, layer 4, and layer 7) are references respectively to the second data link layer, the third network layer, the fourth transport layer, and the seventh application layer of the OSI (Open System Interconnection) layer model.

[0037] **Figure 2** illustrates a manifest-processing framework 200 that processes application manifests received from tenant administrators in an SDDC. Based on this processing, the framework 200 interacts with compute and network managers in the SDDC to deploy a multi-segmented application and to configure the forwarding and service elements in the SDDC to set up the desired communication rules between the segments of this application and between these

segments and other applications and devices inside and outside of the SDDC. In some embodiments, application manifests can also include requests for adjusting previously deployed multi-segmented applications and/or previously configured communication profiles for the previously deployed segments.

[0038] As shown, the manifest framework includes a parser 205, a constraint checker 210, a sorter 220, an orchestrator 230, and several rule and policy storages 215, 225 and 235. The operations of these components will be described by reference to **Figure 3**, which illustrates a process 300 that these components perform for an application manifest. As shown, the process 300 starts when the manifest-processing framework 200 receives (at 305) an application manifest from an administrator's machine 260 (e.g., from a VM, container, or standalone computer used by the administrator to specify the application manifest). As mentioned above and further described below, the administrator in some embodiments can use an application manifest template provided by the manifest framework to specify the manifest 255.

[0039] In some embodiments, the application manifest 255 includes a syntactical representation of a multi-segmented application. An application manifest in some embodiments is a hierarchical API that includes two or more commands that define or modify (1) one or more application segments, and (2) one or more communication profiles between each application segment and another application segment or another application/machine inside or outside of the SDDC.

[0040] The application manifest is a hierarchical API, as different commands can be nested under other commands, e.g., a domain definition can include an application-segment group definition which in turn can include one or more definitions of one or more machines (e.g., specific VMs or containers) to implement an application segment. In some embodiments, the manifest is defined in a declarative language. For instance, the manifest is written in a Javascript Object notation (JSON) format in some embodiments, but in other embodiments it can be expressed in other hierarchical formats such as the XML (Extensible Markup Language) format.

[0041] After receiving an application manifest 255, the parser 205 of the framework 200 identifies (at 310) several different requests (commands) contained in the manifest. For instance, for a typical three-tier application, the manifest can specify the deployment of a webserver, an application server, and a database server. In such a situation, the parser would parse the manifest into three sets of one or more commands, with each set associated with the deployment of one of the tiers (e.g., the deployment of the webserver, appserver, or database server). In some embodiments, the

parser generates an input API tree from the manifest. The input API tree represents the parent child relationship between the different portions of the manifest.

[0042] After the parser breaks the manifest into several individual requests, the constraint checker 210 determines (at 315) whether any of the individual requests violate policy constraints stored in its constraint storage 215. If so, the framework returns an error to the administrator. Otherwise, the sorter 220 identifies (at 320) a sorted order for implementing these requests. To identify this sorted order, the sorter in some embodiments constructs a type specific map that identifies each SD resource identified in the manifest according to its type. To do this, the sorter in some embodiments performs a breadth-first traversal of the input API tree constructed from the manifest by the parser, classifies the input into different buckets based on the resource types, and stores the classified input in the type specific map.

[0043] Each key in the type specific map is a resource type, and the value for each key is a list of all resources of the specific type in the input API tree. Each node element is stored along with its parent. In sum, the input API tree is classified based on resource types, e.g., all domains in one bucket, all groups in another and so on. After generating the type specific map, the sorter defines an execution order for persisting the SD resources in the input API tree. In some embodiments, the execution order is a predefined ordered list of resource types. This list governs the order in which resources in the input tree should be persisted. If a new type is introduced in the system, the execution order is dynamically updated to include the order for the new element. For example, a sample execution order in some embodiment would be (1) Domain, (2) Group, and (3) Communication Map. This means domains should be created first, then the groups, and then communication maps.

[0044] Next, the sorter uses a service provider registry to persist the SD resources in the constructed API tree. The service provider registry is a map of resource types to callback handlers. The callback handlers are registered for each type in the system. The responsibility of a callback handler is to persist the type for which it is registered. As further described below, the callback handlers are implemented by deployment plugins in some embodiments. A deployment plugin is a module that plugs into an API processing system to handle the persistence of changes requested by received APIs and the deployment of the persisted changes.

[0045] Once the sorter identifies the invocation order and, based on this order, invokes the call back handlers to persist the deployment data to one or more configuration databases, the

orchestrator 230 interacts (at 325) with one or more network, compute and/or service managers 240 to deploy the SD resources 250 based on the configuration data that has been persisted to the configuration databases. In some embodiments, the orchestrator 230 is implemented by the deployment plugins that also implemented the callback handlers for persisting the data to the configuration databases.

[0046] Also, in some embodiments, the SDDC resource managers 240 use one or more SDDC resource controllers 245 to deploy the multi-segmented application and its associated communication profiles on the SDDC resources 250. Examples of such resources include host computers, VMs, containers, software and hardware forwarding elements, software and hardware middlebox service elements, etc. The resource managers and controllers 240 and 245 include (1) compute managers and controllers that deploy and configure the application segments of the multi-segment application defined in the manifest, and (2) network managers and controllers that define and deploy the network forwarding and service rules for implementing the communication profiles for the application segments as specified by the manifest.

[0047] In some embodiments, the application segments are deployed as VMs or containers executing on host computers, and/or as standalone computers, in the SDDC. Similarly, the network forwarding and service rules in some embodiments are processed by software forwarding elements (e.g., software switches and routers) and software middlebox service VMs, service containers and/or services modules executing on the host computers in the SDDC. These forwarding and/or service rules are also configured in some embodiments on hardware forwarding elements (e.g., top-of-rack switches), standalone hardware or software gateways, and/or standalone middlebox appliances in the SDDC.

[0048] **Figure 4** illustrates an example of an application manifest that defines a multi-segmented application called Slack. The application manifest in this example is a hierarchical API template 400 that can be used to define an actual application manifest to send to the manifest-processing framework. Such a template provides a mechanism to specify a common set of requests that are often invoked in sequence in order to deploy a multi-segmented application. The template API allows the administrator to deploy the common set of requests without having to define the series of APIs from scratch.

[0049] To specify an actual multi-segment application manifest off of a manifest template 400, an administrator in some embodiments just needs to modify a limited number of fields, called

placeholder fields, in a copy of the template that becomes the manifest. From this perspective, a template API is a set of one or more requests (with respect to one or more resources) with blank fields or placeholders. Examples of placeholder application name, application identifier (App_ID), process name, process hash, user identifier, or any other key value pairs used in the multi-segment application template. In some embodiments, the administrator can also modify other components (e.g., add segments, add or delete communication rules, modify communication rule parameters, etc.) of a copy of template that becomes the actual manifest.

[0050] In some embodiments, the API template is a managed resource. It is represented by a list of placeholders and a body which is the API object. The API template 400 of **Figure 4** is for deploying a Slack application. **Figure 5** illustrates an exemplary deployment of the Slack application 500. This application has a data model that is pictorially illustrated in **Figure 6**. Both the application 500 and the data model 600 will be further described below in order to further describe the portions of the application manifest 400.

[0051] The application manifest 400 is in a hierarchical JSON format, which is equivalent to a tree format. Each node of the tree corresponds to an SDDC resource and has a field that describes the resource type for that node. Each node has a special property that holds all the children for the node depicting the parent-child relationship. Child nodes can in turn have multiple children and this can go to any depth. Thus, each node can be a parent and a child at the same time (similar to a non-leaf node in a tree).

[0052] In **Figure 4**, each node has a property "resource_type" that describes the node's type. Example types in some embodiments include Infra, Tenant, Domain, Group, CommunicationMap, CommunicationEntry, etc. These are all different types of resources in a datacenter. A node can also have a property "Children" that holds all the children of the node. For instance, in **Figure 4**, the node of type Domain 410 is a child of type Infra 405 and has eight children 415-450 of two different types which are "Group" and "CommunicationMap." In some embodiments, Tenant refers to a tenant in a multi-tenant SDDC, Domain is the workload under a tenant, CommunicationMap are security policies, and CommunicationEntry are the rules under a security policy.

[0053] In some embodiments, each SD resource can be identified with a unique path from the root with all taxonomical parents included in the path. For example, a /vmware specifies all resources associated with a tenant VMware. A path/vmware/domains/Outlook specifies all Outlook

workloads for the tenant VMware. A path /vmware/domains/Outlook/communication-maps/web-profile specifies the web-profile of the Outlook workloads of the tenant VMware. A path

/vmware/domains/Outlook/communicationmaps/web-profile/communication-entries/open-browser-access

specifies the open browser access of Outlook workload of the tenant VMware. More generally, the format for the path for a security policy can be specified as: /<tenant-name>/domains/<workload-name>/communication-maps/<security-policy- name>/communication-entries/<rule-name>.

[0054] As shown in **Figure 4**, this manifest template 400 includes a template header 401 that provides a name 402 and a description 403 of the template along with a placeholder list 404. The manifest template 400 also includes eleven requests 405-480. Request 405 is to create a construct called Infra. This construct has eight children as defined by requests 405-450. The request 410 defines a Domain called slack_app.

[0055] Requests 415-445 define seven application segments of the slack_app multi-segment application. These seven segments are illustrated in **Figure 5**. As shown, these seven segments include slack sharing 515 (defined by request 415), slack base 520 (defined by request 420), slack call 525 (defined by request 425), slack editing 530 (defined by request 430), slack downloading 535 (defined by request 435), slack uploading 540 (defined by request 440), and slack file-transfer 545 (defined by request 445). Each request 415-445 for each segment specifies that the segment is to be implemented by a VM that is tagged with a key that is set equal to the name (identifier) for that segment.

[0056] **Figure 5** and **6** show the communications between these segments go through slack base 520. As such, the request 450 defines the communication profile (i.e., Communication Map) between these segments in terms of six communication rules (i.e., Communication Entries) regarding data message exchange between the slack base 520 and each of the other segments 515 and 525-545. In some embodiments, these six communication rules are implemented by firewall rules in the dataplane. **Figure 5** also illustrates that in some embodiments the communication rules (e.g., the firewall rules) can be based on any number of different contextual attributes, such as appID, process name, etc.

[0057] As shown, each communication entry is expressed in terms of the source of a data message (e.g., a source group), a destination of a data message (e.g., a destination group), an action specifying whether the communication is allowed or rejected, and a set of services, ports and/or

protocols that are used by the data message. The Communication Map also defines a rule 480 (i.e., a communication entry) that specifies how the slack base may communication with an active directory service 590, shown in **Figure 5**. In some embodiments, these communication entries 455-480 are be translated into firewall rules by the manifest-processing framework for controlling communication between the different segments of Slack and between other machines (inside or outside of the SDDC) and these segments.

[0058] In some embodiments, a template can be managed through GET, PATCH, DELETE, and POST commands. For instance, in some embodiments, a GET/policy/templates returns a list of template identifiers in the database. A GET /policy/templates/<template-id> in some embodiments returns the template for a specific template identifier. Also, in some embodiments, a PATCH /policy/templates followed by a template JSON definition creates a new template. A DELETE /policy/templates/<template-id> in some embodiments deletes a template given a specific template identifier.

[0059] A POST /policy/templates/<template-id>?action=deploy is specified in some embodiments to define and invoke a hierarchical API based on a template API. This command deploys a template given a specific template identifier <template-id>. Arguments that provide the values of the placeholders in the template will be passed in the body of the POST request. In response to the POST command along with the placeholder arguments, a template manager of the manifest-processing framework in some embodiments fetches the identified template, applies arguments that represent the placeholder values in order to define the hierarchical API, and then creates one or more request objects to identify each requested operation in the hierarchical API. Such a template manager will be further described below.

[0060] In sum, the manifest template 400 specifies the set of processes that define the multi-segment application and also the set of recommended communication profiles. These definitions can be modified by the administrator based on the administrator's requirements. In other words, the administrator has the option to use this verbatim in the environment or make modifications as deemed necessary in their data center. This sample application manifest for Slack can be published as a standard across the industry for easier deployment and micro-segmentation of this application.

[0061] Figure 7 illustrates a process 700 that represents an exemplary flow for defining and deploying a multi-segmented application based on a manifest template. As shown, the process 700 initially provides (at 705) a list of application manifest templates. The manifest templates in some

embodiments are templates for the most commonly used multi-segmented applications. The manifest templates in some embodiments are open sourced and community driven so that there is more exposure and accuracy for these applications.

[0062] Next, at 710, an administrator selects a manifest template from the list of provided manifest templates. The selected template is for the multi-segment application that the administrator wants to deploy. At 715, the administrator verifies the manifest based on his intent. The selected manifest shows the compute, networking, and security intent using the default configuration as defined by the publisher of the template. The administrator has the option to accept the default configuration or modify it to match with the desired intent.

[0063] At 720, the administrator submits the manifest to the manifest-processing framework for processing and deployment. Once the manifest is published to the framework, the framework deploys (at 725) the compute, network, and/or service resources specified in the manifest. After 725, the process 700 ends.

[0064] In some embodiments, the framework uses a new classification engine, which is a middlebox service the collects different types of contextual attributes from the deployment environment (such as those used by existing or new workloads) so that these attributes can be used by administrators in defining the communication entries in the manifest. **Figure 8** illustrates the architecture of this new classification engine for some embodiments.

[0065] Applications running on the VMs are not ephemeral by nature. Once an administrator installs an application, these applications typically run for a long time. This applies to VMs and bare metal servers typically. In some embodiments, context engines running on hypervisors executing on host computers periodically send to the SDDC management plane lists of running processes on their guest VMs. The Application discovery engine 805 receives this information and provides a visualization about the application information and the associated virtual machine.

[0066] In some embodiments, this information is not polled continuously, and the process is not automated. Accordingly, the new classification vertical on management plane in some embodiments performs a periodic synchronization operation internally to detect the list of running/installed processes on the VMs and tag them based on the process information. The inventory vertical 810 gets the list of VMs in the system and creates a security group internally and enables application discovery for these VMs. Once the list of applications/processes running on the VMs are identified, the VMs are tagged with the process. The grouping/tagging manager

815 collects the security groups and other tagging data for the classification engine 800, while the firewall manager 820 collects the contextual data from and for the deployed firewalls. Based on the collected tags, the intent from the policy created at the application level determines the security groups and the communication profile for these processes.

[0067] **Figure 9** illustrates an example of the API processing system 900 of some embodiments of the invention. This API processing system 900 implements the manifest-processing framework 200 of some embodiments. In this system, each tenant can create an SDDC cluster 902 that includes one or more SDDC instances 905, which can be considered to be separate environments. As shown, each SDDC instance 905 in some embodiments includes an API gateway 920, an API processor 925, a compute manager 910, a network manager 915, a controller 940, a template manager 927, a policy checker 923, a configuration data storage 935, and several deployment plugins 930.

[0068] In some embodiments, two or more of these components execute on two or more machines (e.g., VMs, containers, standalone servers, etc.) in one or more datacenters and communicate with each other through a network. In these or other embodiments, each SDDC instance includes multiple instances of each of these components for distributing the load and for high availability.

[0069] The compute manager 910 deploys and manages workload machines (e.g., workload VMs or containers). The network manager 915 on the other hand deploys network resources (e.g., software switches and routers) and middlebox service resources (e.g., service VMs and modules) in a datacenter. In some embodiments, the compute and network managers 910 and 915 use one or more controllers 940 to distribute the configuration data that is stored in one or more configuration data storages 935 to host computers, forwarding elements (e.g., software switches and routers executing on host computers, or standalone switches and routers), service machines (e.g., service VMs, service containers, other service modules, and standalone service appliances), and other resources in the SDDC.

[0070] The API gateway 920 redirects all API commands to the API service module 925, or to the UI manager 922 in some cases, based on URL patterns. The UI manager 922 processes API commands that are received through a graphical user interface and directs these commands to the API processor 925. The API processor 925 performs the processes illustrated in **Figures 2 and 3** to ensure that the different requests that are part of a received application manifest are persisted to the configuration data storage(s) 935 and deployed in the correct order. The API processor 925

owns the user's desired state that it stores in its data storage 932. In some embodiments, the API processor 925 runs as a VM or container.

[0071] As shown, the API processor 925 in some embodiments uses the template manager 927, which has access to several manifest templates 929 that specify multi-segment application configurations for the SDDC resources. Through the template manager 927, users can select and modify a template (e.g., through API commands) to produce a complete manifest. Based on this completed manifest, the API processor 925 can then deploy, or update a previously deployed, set of SDDC resource to deploy or adjust a previously deployed multi-segment application.

[0072] To deploy a resource, or update a previously deployed resource, based on a request in a received manifest or a manifest completed through the invocation of a manifest template with the required input, the API processor 925 in some embodiments parses the manifest into one or more requests and uses the policy checking engine 923 to validate each request (i.e., to specify whether each request satisfies the constraints specified in the policies that are stored in the policy storage 924 and that are applicable to the resources referred to in the request).

[0073] In some embodiments, each policy in the policy storage 924 includes (1) a target that specifies a set of one or more datacenter resources to which the policy applies, and (2) an expression that specifies a constraint on operations on the specified resource set. The policies are expressed in a declarative format in some embodiments. Thus, for each request in a manifest, the policy engine compares a set of attributes of the selected request's resource with a policy's target to determine whether the policy is applicable to the resource. After identifying one applicable policy, the policy checking engine determines whether the identified policy's expression specifies a constraint that requires the selected request to be rejected or allowed.

[0074] Through the deployment plugins 930, the API processor 925 persists the SD resource data in the API calls in the configuration database 935. The deployment plugins 930 run as VMs or containers in some embodiments. Each plugin 930 is responsible for deploying one or more SD resource types. Examples of such types include data compute nodes (e.g., compute machines such as VMs or containers), distributed firewall rules, edge firewall rules, L2 and L3 forwarding elements (software switches and routers), security groups, VPN services, DHCP services, DNS services, load balancing services, etc.

[0075] To deploy these services, the plugins 930 interact with the compute manager 910 and the network manager 915, which, in turn, interact with one or more controllers 940. Through these

managers and controllers, the plugins 930 distribute configuration data from the persistent database 935 to host computers and standalone network/service devices in an SDDC in order to direct these computers and devices to deploy the desired SD resources.

[0076] In some embodiments, there is one desired state and orchestration service (i.e., API processing module) per SDDC instance. This is a highly available service that is deployed in some embodiments in the form of a container or a VM. This service accepts user's intent and performs orchestration across different services. This service also owns the details of the enforcement points (compute and network managers) to which the policies need to be pushed down.

[0077] The deployment plugins 930 provide the realization of the intent. As mentioned above, each of these plugins in some embodiments is deployed as a separate service running in a separate container or VM. In some embodiments, some services are packaged together in a single container, but run as separate services in terms of design and communication. Since the orchestration is performed by the desired state service, each of the plugin services in some embodiments exposes a set of REST API endpoints that would be invoked. Also, the desired state service in some embodiments serves as the common service that returns the state of the realized resources across different services. This is the case even though in some embodiments the realized state data is updated in the data store by the plugin services.

[0078] Thus, execution of manifest results in creation of desired state in one go. If the system is able to validate and persist whole of intent, a notification is sent (e.g., an http status code 200 OK is returned) to the source of the manifest. After the intent is created, notifications are generated. These notifications are consumed asynchronously by deployment plugins. The deployment plugins then take care of realizing the intent. Status of realization can be queried from the system using status APIs

[0079] The API processing system 900 in some embodiments provides users with the ability to query the intent in hierarchical fashion. For instance, in some embodiments, the system provides a GET API that facilitates the reading of entire intent in one go. A special flag is passed in URL parameter to request GET in hierarchical fashion. When the parameter is not passed, the GET in some embodiment would work as normal GET and a single resource is returned. Hierarchical GET in some embodiments can work on the entire tree or the parts of the tree, i.e. it can specify the node from which the hierarchy is to be retrieved, as hierarchical GET can work from any level within the tree.

[0080] Another aspect of hierarchical GET is filtering in some embodiments. An administrator in these embodiments can filter out the intent tree to see only the types that she is interested in. This filtering can be a simple type-based filtering, e.g., the administrator can say GET the intent hierarchy for type "Domain". In an advanced filtering mechanism, user can choose to retrieve intent based on features, e.g. the administrator can say GET all the resources in intent hierarchy related to firewall feature.

[0081] In some embodiments, user can perform hierarchical GET and club it with hierarchical POST. The administrator in some embodiments can retrieve the intent hierarchy, then modify and POST it back. This enables "import/export" use case. The administrator in some embodiments can also retrieve a manifest and store it. Subsequently, the administrator can restore the previously retrieved intent.

[0082] **Figure 10** illustrates how some embodiments enforce context-based firewall rules on host computers. In some embodiments, the SDDC achieves its desired segmentation communication profiles by defining and enforcing such context-based rules on host computers. As mentioned above, the manifest-processing framework translates the communication profile rules defined in the manifest which are converted to rules that the network management layer (i.e., the network manager cluster) can process to achieve the required micro-segmentation for the application.

[0083] **Figure 10** shows the network management cluster receiving a set of rules that map these application segments defined in the manifest to security groups based on process names/hash by using the grouping provider. The network management cluster also receives the translated firewall rules matching these process-based groups based on the communication profiles defined in the manifest. Once the process based groups and the firewall rules are created, firewall managers 1010 of the manager cluster 1005 distribute the rules to the host computers and other enforcement nodes in the SDDC.

[0084] To map files, modules, and other process data to a network event, some embodiments use contextual data captured through guest introspection (GI) agents 1020 executing on machines 1015 (e.g., host VMs and containers). GI agents 1020 in some embodiments capture any guest network connection and file access and its associated process context. In some embodiments, all network connections and file access are intercepted by this agent and sent to the context service engine 1030 running on the host. In the ESX hypervisor provided by Vmware Inc., the contextual data is sent to the context engine 1030 through a hypervisor component called a Mux 1025, which acts as

a conduit to send the network and file events from the GI agents to the context engine executing on the host computer. Using this information, the context engine updates the context table 1035 that contains information about the user, source IP address, source port, protocol, and the process information that are initiated by the guest VM.

[0085] When the data compute machine 1015 subsequently makes a network connection, a firewall module 1040 executing on the host computer inspects the outgoing packet, associates the process information with the packet flow from the context table, and matches it with the rules in the firewall table thereby enforcing the effective firewall rules at the process level. This level of fine grained control of the processes running on the VM's helps the administrator detect unsanctioned software/applications installed in the environment and further block malicious processes running in the system. The effective rule realization is communicated back to the network management cluster, which provides the status of the application deployed and the effective rules to the administrator. Context based firewalls and other middlebox service engines are further described in U.S. Patent Application 15/650,251, which is published as 2018-0181423 and incorporated herein by reference.

[0086] The manifest-processing framework in some embodiments has an intent based learning engine. Using the GI agent installed on the guest machines (e.g., guest VMs or containers), the manifest-processing framework can not only identify the processes involved in the detected network activities but also the installed binaries and the file attributes associated with these activities. Some of the most common attributes includes file hash, publisher information, installation path, and the certificate. In some embodiments, these identified processes and other attributes are collected from the deployment environment (e.g., from the host computers) and analyzed by the learning engine, in order to produce new multi-segment application templates. The collected and analyzed data shows how some administrators typically deploy multi-segment applications (e.g., new multi-segment applications). The learning engine in some embodiments discerns the common deployment attributes from the collected data, and based on its analysis, recommends new multi-segment manifest templates.

[0087] As mentioned above, the network and file events captured by the GI agents are stored in the context table. From here, the manifest-processing framework can collect data regarding these captured events as shown in **Figure 11**. In some embodiments, this data is collected in a separate server cluster or set of appliances, as the data can grow very large depending on the collection time

and the application workload. This information is stored at a host level about all the processes seen on each VM in the host.

[0088] In some embodiment, the collected network and file event data is then aggregated and modeled to generate application templates customized to obtain the right segmentation at a process level within a VM. For each network event, the GI agent can get the corresponding process, socket, rate of connections, and type of connection. The framework in some embodiments has the communication channels established to query the libraries used by the process at that instance and the files which are accessed by the process.

[0089] This data in addition with round trip time, time of the day, and correlation with other processes is used in some embodiments to train an unsupervised or semi supervised machine learning model. Some embodiments use a One-Class Support Vector Machine (SVM) classifier for identifying the process information. The SVM module is useful in scenarios where there is a lot of "normal" data, and there are not many cases of the anomalies that need to be detected.

[0090] One-class SVM is an unsupervised algorithm that learns a decision function for novelty detection: classifying new data as similar or different to the training set. SVMs are supervised learning models that can be used for classification tasks. Typically, an SVM is given labelled data that can be mapped to an n-d space. The different categories are divided by a clear gap, and the SVM finds the one that is as wide as possible. The new data samples are classified belonging to a category based on which side of the gap they fall in. Some embodiments classify or process information based on new data seen and recommend the right set of application templates that the administrator needs to protect their network. Using this model, some embodiments are able to predict the running applications and their corresponding processes. In this manner, the model helps identify new applications and recommend the right application template to secure the application.

[0091] The above-described embodiments offer several advantages. They provide an intent-based API processing system for deploying multi-segmented applications through a hierarchical API data model that allows the users to specify their intent (e.g., several application segments and the communication profiles between them) without worrying about the mechanics of persisting and realizing these resources. In some embodiments, the intent-based API system allows the users to define hierarchical API commands by using a simple declarative language that refers to a simplified hierarchical data model. Each hierarchical API command can define multiple SD resources at multiple resource levels in the SDDC without requiring earlier API commands to

create certain SD resources before others. In fact, one hierarchical command can be used in some embodiments to define all the SD resources for one user (e.g., one tenant) of the SDDC (e.g., of a multi-tenant SDDC).

[0092] The manifest-processing framework of some embodiments leverages the hierarchy of the data model to provide a process for accepting, validating, and realizing parts or the whole of the hierarchy in a single API invocation. This system leverages the inherent knowledge of the data model to identify dependencies and invoke the underlying services in the right order, both for persistence and realization of the intent. Also, all of the persistence is done in a single transaction, thus making sure that the entire intent is accepted as an atomic unit.

[0093] Once the manifest-processing framework determines that the multi-segment application defined in the received manifest is deployable, the API system uses an asynchronous process in some embodiments to deploy resources automatically in the right order without further input from the administrator. This process works with one or more network, service, or compute managers to deploy or update one or more network, service, or compute resources based on a work order that is appropriate for the deployed resources.

[0094] Many of the above-described features and applications are implemented as software processes that are specified as a set of instructions recorded on a computer readable storage medium (also referred to as computer readable medium). When these instructions are executed by one or more processing unit(s) (e.g., one or more processors, cores of processors, or other processing units), they cause the processing unit(s) to perform the actions indicated in the instructions. Examples of computer readable media include, but are not limited to, CD-ROMs, flash drives, RAM chips, hard drives, EPROMs, etc. The computer readable media does not include carrier waves and electronic signals passing wirelessly or over wired connections.

[0095] In this specification, the term "software" is meant to include firmware residing in read-only memory or applications stored in magnetic storage, which can be read into memory for processing by a processor. Also, in some embodiments, multiple software inventions can be implemented as sub-parts of a larger program while remaining distinct software inventions. In some embodiments, multiple software inventions can also be implemented as separate programs. Finally, any combination of separate programs that together implement a software invention described here is within the scope of the invention. In some embodiments, the software programs,

when installed to operate on one or more electronic systems, define one or more specific machine implementations that execute and perform the operations of the software programs.

[0096] Figure 12 conceptually illustrates a computer system 1200 with which some embodiments of the invention are implemented. The computer system 1200 can be used to implement any of the above-described hosts, controllers, and managers. As such, it can be used to execute any of the above described processes. This computer system includes various types of non-transitory machine readable media and interfaces for various other types of machine readable media. Computer system 1200 includes a bus 1205, processing unit(s) 1210, a system memory 1225, a read-only memory 1230, a permanent storage device 1235, input devices 1240, and output devices 1245.

[0097] The bus 1205 collectively represents all system, peripheral, and chipset buses that communicatively connect the numerous internal devices of the computer system 1200. For instance, the bus 1205 communicatively connects the processing unit(s) 1210 with the read-only memory 1230, the system memory 1225, and the permanent storage device 1235.

[0098] From these various memory units, the processing unit(s) 1210 retrieve instructions to execute and data to process in order to execute the processes of the invention. The processing unit(s) may be a single processor or a multi-core processor in different embodiments. The read-only-memory (ROM) 1230 stores static data and instructions that are needed by the processing unit(s) 1210 and other modules of the computer system. The permanent storage device 1235, on the other hand, is a read-and-write memory device. This device is a non-volatile memory unit that stores instructions and data even when the computer system 1200 is off. Some embodiments of the invention use a mass-storage device (such as a magnetic or optical disk and its corresponding disk drive) as the permanent storage device 1235.

[0099] Other embodiments use a removable storage device (such as a floppy disk, flash drive, etc.) as the permanent storage device. Like the permanent storage device 1235, the system memory 1225 is a read-and-write memory device. However, unlike storage device 1235, the system memory is a volatile read-and-write memory, such as random access memory. The system memory stores some of the instructions and data that the processor needs at runtime. In some embodiments, the invention's processes are stored in the system memory 1225, the permanent storage device 1235, and/or the read-only memory 1230. From these various memory units, the processing unit(s) 1210 retrieve instructions to execute and data to process in order to execute the processes of some embodiments.

[00100] The bus 1205 also connects to the input and output devices 1240 and 1245. The input devices enable the user to communicate information and select commands to the computer system. The input devices 1240 include alphanumeric keyboards and pointing devices (also called "cursor control devices"). The output devices 1245 display images generated by the computer system. The output devices include printers and display devices, such as cathode ray tubes (CRT) or liquid crystal displays (LCD). Some embodiments include devices such as touchscreens that function as both input and output devices.

[00101] Finally, as shown in **Figure 12**, bus 1205 also couples computer system 1200 to a network 1265 through a network adapter (not shown). In this manner, the computer can be a part of a network of computers (such as a local area network ("LAN"), a wide area network ("WAN"), or an Intranet), or a network of networks (such as the Internet). Any or all components of computer system 1200 may be used in conjunction with the invention.

[00102] Some embodiments include electronic components, such as microprocessors, storage and memory that store computer program instructions in a machine-readable or computer-readable medium (alternatively referred to as computer-readable storage media, machine-readable media, or machine-readable storage media). Some examples of such computer-readable media include RAM, ROM, read-only compact discs (CD-ROM), recordable compact discs (CD-R), rewritable compact discs (CD-RW), read-only digital versatile discs (e.g., DVD-ROM, dual-layer DVD-ROM), a variety of recordable/rewritable DVDs (e.g., DVD-RAM, DVD-RW, DVD+RW, etc.), flash memory (e.g., SD cards, mini-SD cards, micro-SD cards, etc.), magnetic and/or solid state hard drives, read-only and recordable Blu-Ray® discs, ultra-density optical discs, any other optical or magnetic media, and floppy disks. The computer-readable media may store a computer program that is executable by at least one processing unit and includes sets of instructions for performing various operations. Examples of computer programs or computer code include machine code, such as is produced by a compiler, and files including higher-level code that are executed by a computer, an electronic component, or a microprocessor using an interpreter.

[00103] While the above discussion primarily refers to microprocessor or multi-core processors that execute software, some embodiments are performed by one or more integrated circuits, such as application specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). In some embodiments, such integrated circuits execute instructions that are stored on the circuit itself.

[00104] As used in this specification, the terms "computer", "server", "processor", and "memory" all refer to electronic or other technological devices. These terms exclude people or groups of people. For the purposes of the specification, the terms "display" or "displaying" mean displaying on an electronic device. As used in this specification, the terms "computer readable medium," "computer readable media," and "machine readable medium" are entirely restricted to tangible, physical objects that store information in a form that is readable by a computer. These terms exclude any wireless signals, wired download signals, and any other ephemeral or transitory signals.

[00105] While the invention has been described with reference to numerous specific details, one of ordinary skill in the art will recognize that the invention can be embodied in other specific forms without departing from the spirit of the invention. Thus, one of ordinary skill in the art would understand that the invention is not to be limited by the foregoing illustrative details, but rather is to be defined by the appended claims

CLAIMS

1. A method of deploying a multi-segmented application in a software defined datacenter (SDDC), the method comprising:

receiving a hierarchical API command that specifies a plurality of application segments and defines a plurality of rules to control data message flows associated with the application segments;

parsing the API command to identify a set of application segments and a set of rules to deploy; and

based on the parsed API command, deploying a set of machines in the SDDC to implement the set of application segments and providing the set of rules to a set of network elements in the SDDC to control the data message flows associated with the application segments.

2. The method of claim 1, wherein the network elements include managed forwarding elements for forwarding data messages between the application segments and between other machines and the application segments.

3. The method of claim 1, wherein the network elements include middlebox service elements for performing middlebox service operations on data messages sent to or from the application segments.

4. The method of claim 3, wherein the middlebox service elements include middlebox service machines executing on host computers.

5. The method of claim 3, wherein the middlebox service elements include middlebox service filter executing on host computers.

6. The method of claim 3, wherein the service operations include at least one of firewall operations, load balancing operations, network address translation operations, encryption operations, intrusion detection operations, and intrusion prevention operations.

7. The method of claim 1, wherein the middlebox service elements include firewall modules or devices.

8. The method of claim 1, wherein the multi-segmented application has more at least three application segments defined in the hierarchical API command.

9. The method of claim 1, wherein the multi-segmented application has more than five application segments defined in the hierarchical API command.

10. The method of claim 1, wherein the set of deployed machines comprise virtual machines.

11. The method of claim 1, wherein the set of deployed machines comprise containers.

12. The method of claim 1, wherein

the API command includes a set of parameters to update an earlier deployed multi-segment application, and

deploying the set of machines comprises updating the earlier deployed multi-segment application based on a set of parameters specified in the parsed API command.

13. The method of claim 1, wherein

the API command includes a set of parameters to update an earlier provided set of rules, and

providing the set of rules comprises updating the earlier provided set of rules based on a set of parameters specified in the parsed API command.

14. The method of claim 1, wherein at least a subset of rules relates to data message flows between application segments.

15. The method of claim 1, wherein at least a subset of rules relates to data message flows sent to the application segment from applications other than the multi-segment application.

16. The method of claim 1, wherein at least a subset of rules relates to data message flows sent from the application segment to applications other than the multi-segment application.

17. A method for defining a multi-segmented application in a software defined datacenter (SDDC), the method comprising:

creating a hierarchical API command that specifies a plurality of application segments and defines a plurality of rules to control data message flows associated with the application segments;

storing the hierarchical API as a template for the multi-segment application; and

providing an interface to allow the template to be retrieved and customized in order to define a multi-segment application manifest to process to deploy a set of machines in the SDDC to implement the set of application segments and to provide the set of rules to a set of network elements in the SDDC to control the data message flows associated with the application segments.

18. The method of claim 17, wherein the network elements include managed forwarding elements for forwarding data messages between the application segments and between the application segments and applications other than the multi-segment application.

19. The method of claim 17, wherein the network elements include middlebox service elements for performing middlebox service operations on data messages sent to or from the application segments.

20. The method of claim 19, wherein the middlebox service elements include middlebox service machines executing on host computers.

21. The method of claim 19, wherein the middlebox service elements include middlebox service filters executing on host computers.

22. The method of claim 19, wherein service operations include one of a firewall operation, a load balancing operation, a network address translation operation, an encryption operation, an intrusion detection operation, and an intrusion prevention operation.

23. The method of claim 17, wherein the middlebox service elements include firewalls machines or devices.

24. The method of claim 17, wherein the multi-segmented application has more at least three application segments defined in the hierarchical API command.

25. The method of claim 17, wherein the multi-segmented application has more than five application segments defined in the hierarchical API command.

26. The method of claim 17, wherein the set of deployed machines comprise virtual machines or containers.

27. A machine readable medium storing a program which when implemented by at least one processing unit implements the method according to any one of claims 1-26.

28. A computing device comprising:

a set of processing units; and

a machine readable medium storing a program which when implemented by at least one of the processing units implements the method according to any one of claims 1-26.

29. A system comprising means for implementing the method according to any one of claims 1-26.

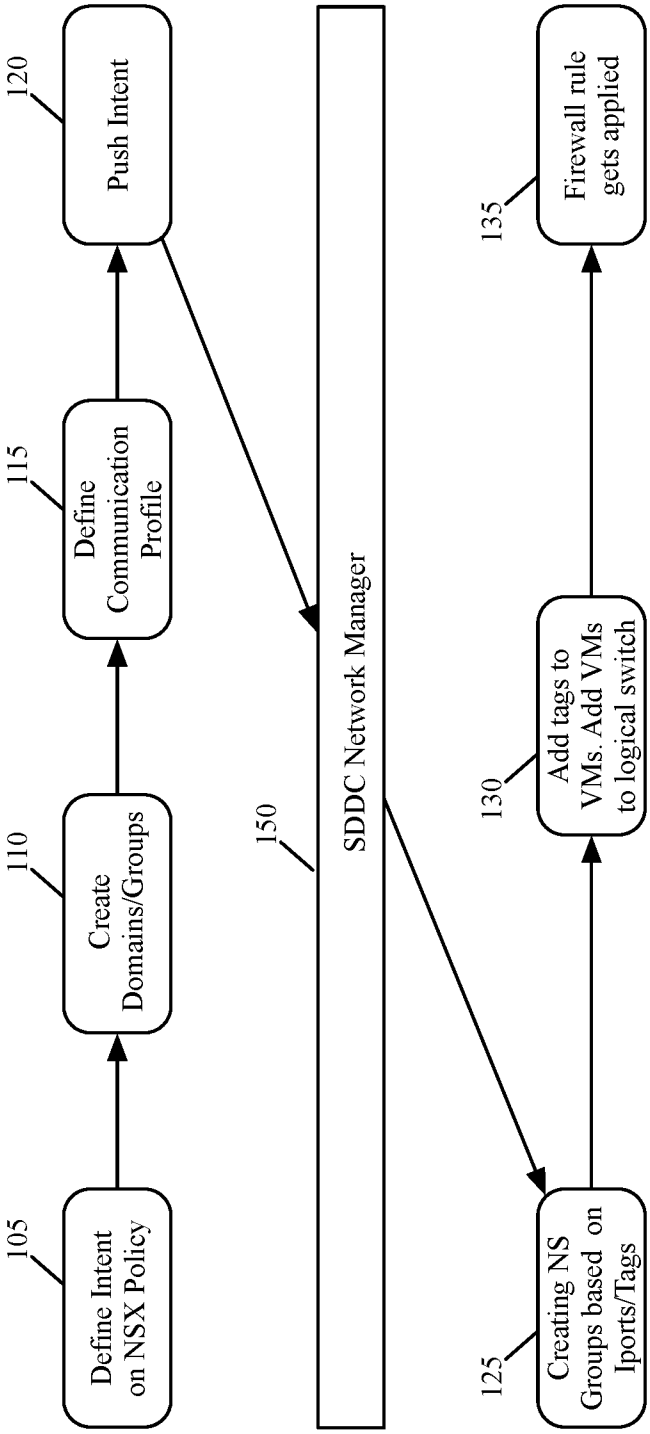


Figure 1
Prior Art

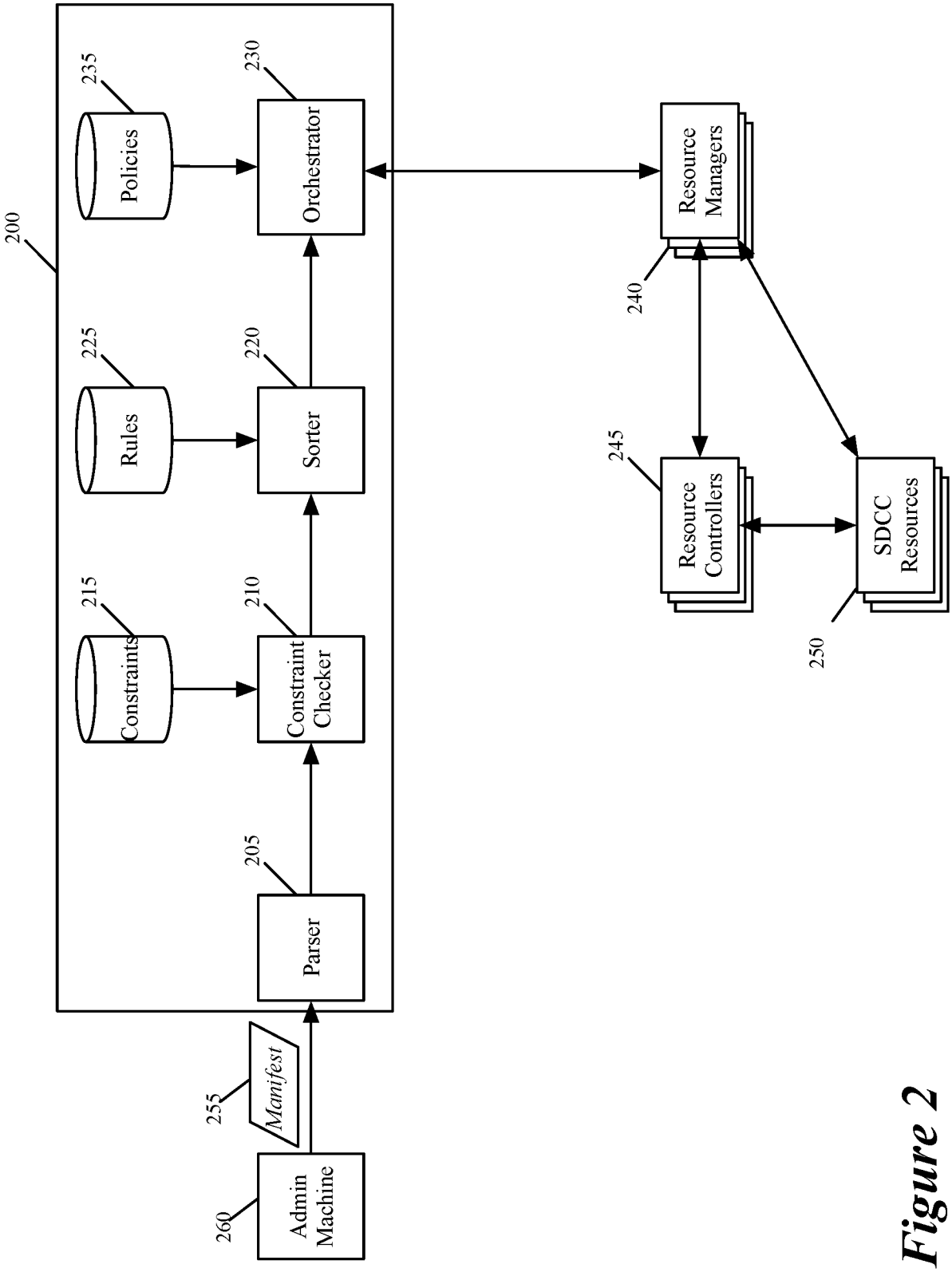

200
Policies
235
Orchestrator
230
Rules
225
Sorter
220
Constraints
215
Constraint Checker
210
Parser
205
Manifest
255
Admin Machine
260
Resource Managers
240
Resource Controllers
245
SDCC Resources
250

Figure 2

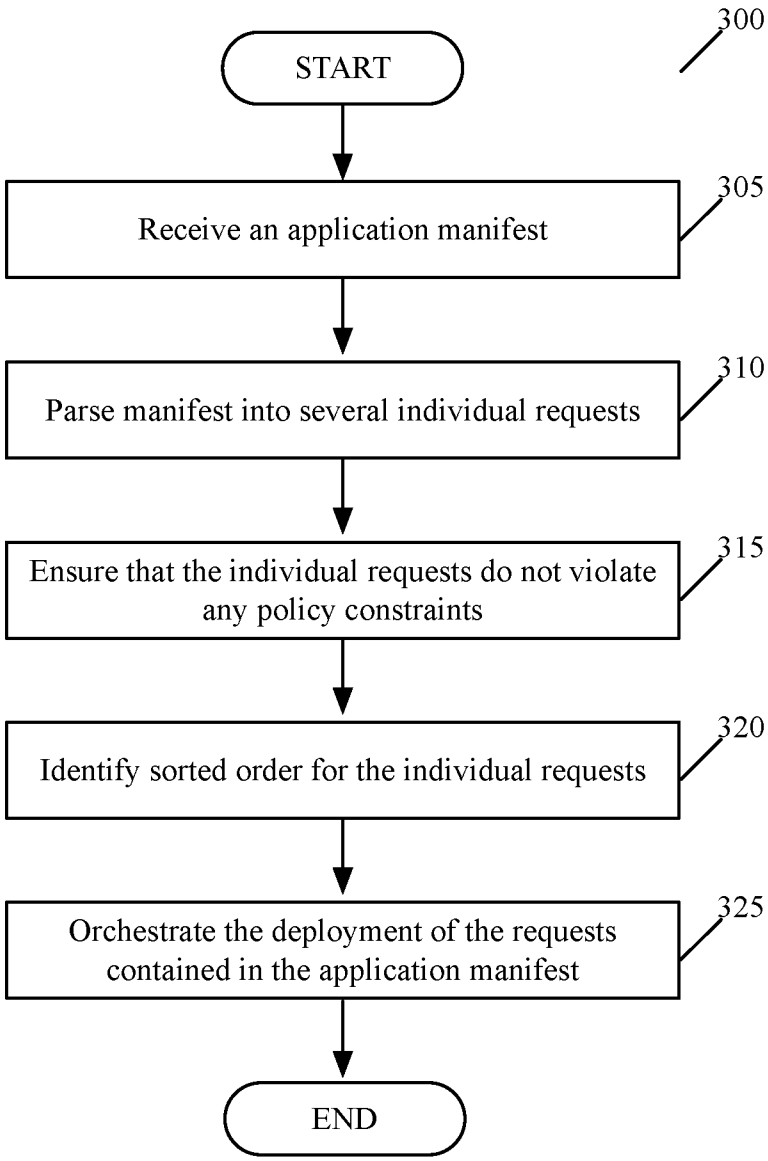

300
START
Receive an application manifest
305
Parse manifest into several individual requests
310
Ensure that the individual requests do not violate any policy constraints
315
Identify sorted order for the individual requests
320
Orchestrate the deployment of the requests contained in the application manifest
325
END

Figure 3



Figure 4:

Figure 4A
Figure 4B
Figure 4C
Figure 4D
Figure 4E

Figure 4A

```
{                                                    420
    "resource_type":"ChildGroup",
    "Group":{
        "_revision":0,
        "id":"slack-base",
        "resource_type":"Group",
        "description":"Policy Group",
        "expression":[
            {
                "resource_type":"Condition",
                "member_type":"VirtualMachine",
                "key":"Tag",
                "operator":"CONTAINS",
                "value":"slack-base"
            }
        ]
    }                                                425
},                                                   
{
    "resource_type":"ChildGroup",
    "Group":{
        "_revision":0,
        "id":"slack-call",
        "resource_type":"Group",
        "description":"Policy Group",
        "expression":[
            {
                "resource_type":"Condition",
                "member_type":"VirtualMachine",
                "key":"Tag",
                "operator":"CONTAINS",
                "value":"slack-call"
            }
        ]
    }                                                430
},                                                   
{
    "resource_type":"ChildGroup",
    "Group":{
        "_revision":0,
        "id":"slack-editing",
        "resource_type":"Group",
        "description":"Policy Group",
        "expression":[
            {
                "resource_type":"Condition",
                "member_type":"VirtualMachine",
                "key":"Tag",
                "operator":"CONTAINS",
                "value":"slack-editing"
            }
        ]
    }
},
```

Figure 4B

```
{                                                          435
    "resource_type":"ChildGroup",
    "Group":{
        "_revision":0,
        "id":"slack-downloading",
        "resource_type":"Group",
        "description":"Policy Group",
        "expression":[
            {
                "resource_type":"Condition",
                "member_type":"VirtualMachine",
                "key":"Tag",
                "operator":"CONTAINS",
                "value":"slack-downloading"
            }
        ]
    }
},                                                         440
{
    "resource_type":"ChildGroup",
    "Group":{
        "_revision":0,
        "id":"slack-uploading",
        "resource_type":"Group",
        "description":"Policy Group",
        "expression":[
            {
                "resource_type":"Condition",
                "member_type":"VirtualMachine",
                "key":"Tag",
                "operator":"CONTAINS",
                "value":"slack-uploading"
            }
        ]
    }
},                                                         445
{
    "resource_type":"ChildGroup",
    "Group":{
        "_revision":0,
        "id":"slack-file-transfer",
        "resource_type":"Group",
        "description":"Policy Group",
        "expression":[
            {
                "resource_type":"Condition",
                "member_type":"VirtualMachine",
                "key":"Tag",
                "operator":"CONTAINS",
                "value":"slack-file-transfer"
            }
        ]
    }
},
```

Figure 4C

```
                                                                    450
{
    "resource_type":"ChildCommunicationMap",
    "CommunicationMap":{
        "_revision":0,
        "resource_type":"CommunicationMap",
        "id":"communication_map",
        "description":"Policy Communication Map",
        "precedence":1,
        "communication_entries":[
                                                                    455
            {
                "_revision":0,
                "resource_type":"CommunicationEntry",
                "id":"slack_sharing",
                "source_groups":["/infra/domains/SlackApplicationDomain/groups/slack-sharing"],
                "destination_groups":["/infra/domains/SlackApplicationDomain/groups/slack-
base"],
                "services":[
                    "/infra/services/HTTPS",
                    "/infra/service/HTTP"
                ],
                "action":"ALLOW"
            },
                                                                    460
            {
                "_revision":0,
                "resource_type":"CommunicationEntry",
                "id":"slack-call",
                "source_groups":["/infra/domains/SlackApplicationDomain/groups/slack-call"],
                "destination_groups":["/infra/domains/SlackApplicationDomain/groups/slack-
base","ANY"],
                "services":[
                    "/infra/services/HTTPS"
                ],
                "action":"ALLOW"
            },
                                                                    465
            {
                "_revision":0,
                "resource_type":"CommunicationEntry",
                "id":"slack-editing",
                "source_groups":["/infra/domains/SlackApplicationDomain/groups/slack-editing"],
                "destination_groups":["/infra/domains/SlackApplicationDomain/groups/slack-
base"],
                "services":[
                    "/infra/services/HTTPS"
                ],
                "action":"ALLOW"
            },
```

Figure 4D

```
                                                                    470
                        {
                            "_revision":0,
                            "resource_type":"CommunicationEntry",
                            "id":"slack-file-sharing",
                            "source_groups":["/infra/domains/SlackApplicationDomain/groups/slack-file-
sharing"],
                            "destination_groups":["/infra/domains/SlackApplicationDomain/groups/slack-
base"],
                            "services":[
                                "/infra/services/HTTPS"
                            ],
                            "action":"ALLOW"
                        }                                                   475
                        ,{
                            "_revision":0,
                            "resource_type":"CommunicationEntry",
                            "id":"slack-downloading",
                            "source_groups":["/infra/domains/SlackApplicationDomain/groups/slack-
downloading"],
                            "destination_groups":["/infra/domains/SlackApplicationDomain/groups/slack-
base"],
                            "services":[
                                "/infra/services/HTTPS"
                            ],
                            "action":"ALLOW"
                        },                                                  480
                        {
                            "_revision":0,
                            "resource_type":"CommunicationEntry",
                            "id":"slack-base",
                            "source_groups":["/infra/domains/SlackApplicationDomain/groups/slack-base"],
                            "destination_groups":["/infra/domains/SlackApplicationDomain/groups/
activedirectory"],
                            "services":[
                                "/infra/services/ldap",
                            ],
                            "action":"ALLOW"
                        }
                    ]
                }
            }
        ]
    }
}
```

Figure 4E

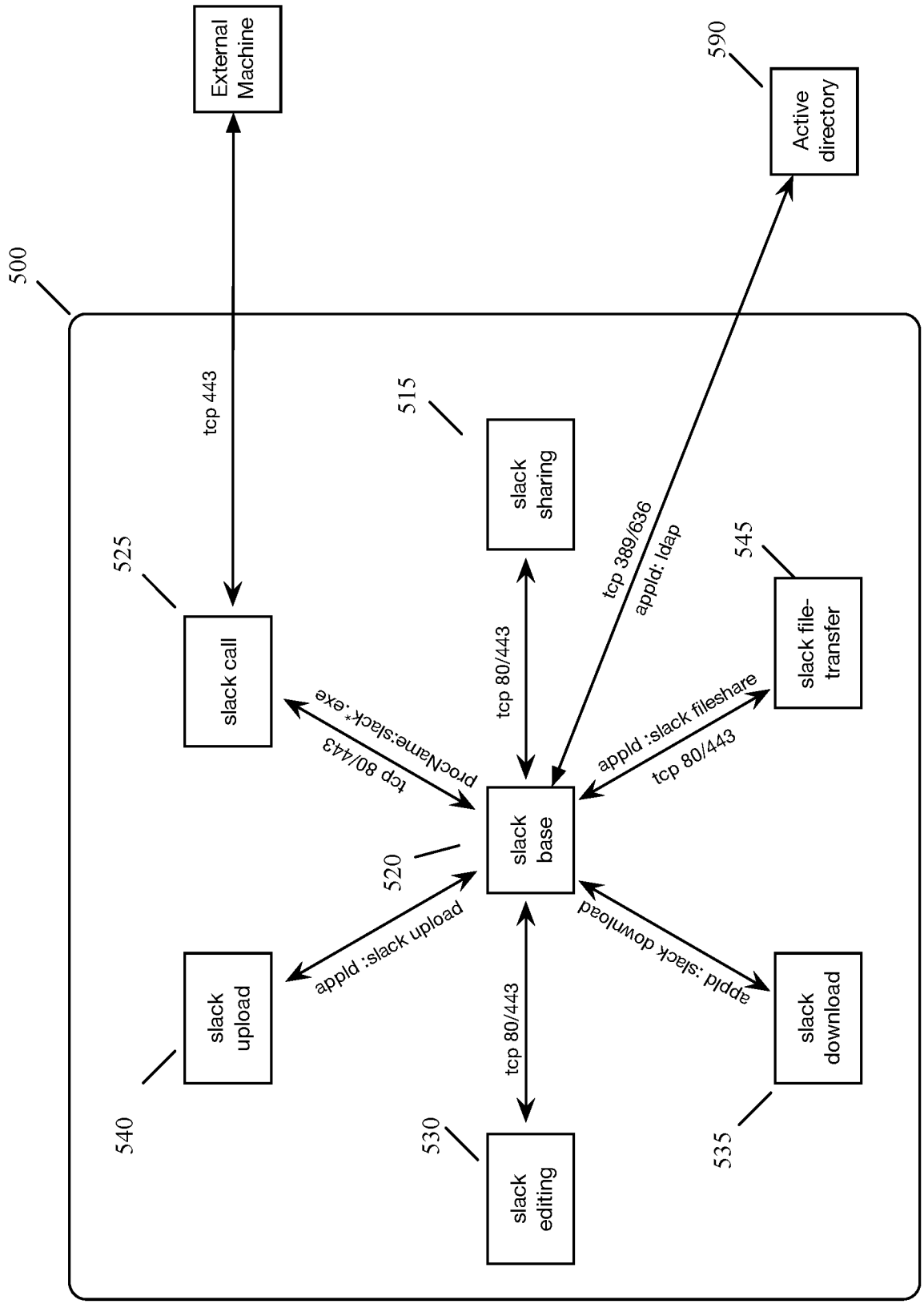

500
External Machine
590
Active directory
tcp 443
515
slack sharing
525
slack call
545
slack file-transfer
tcp 80/443
tcp 389/636
appId: ldap
ProcName:slack*.exe
tcp 80/443
appId :slack fileshare
tcp 80/443
slack base
520
appId :slack upload
appId :slack download
slack upload
slack download
540
535
tcp 80/443
530
slack editing

Figure 5

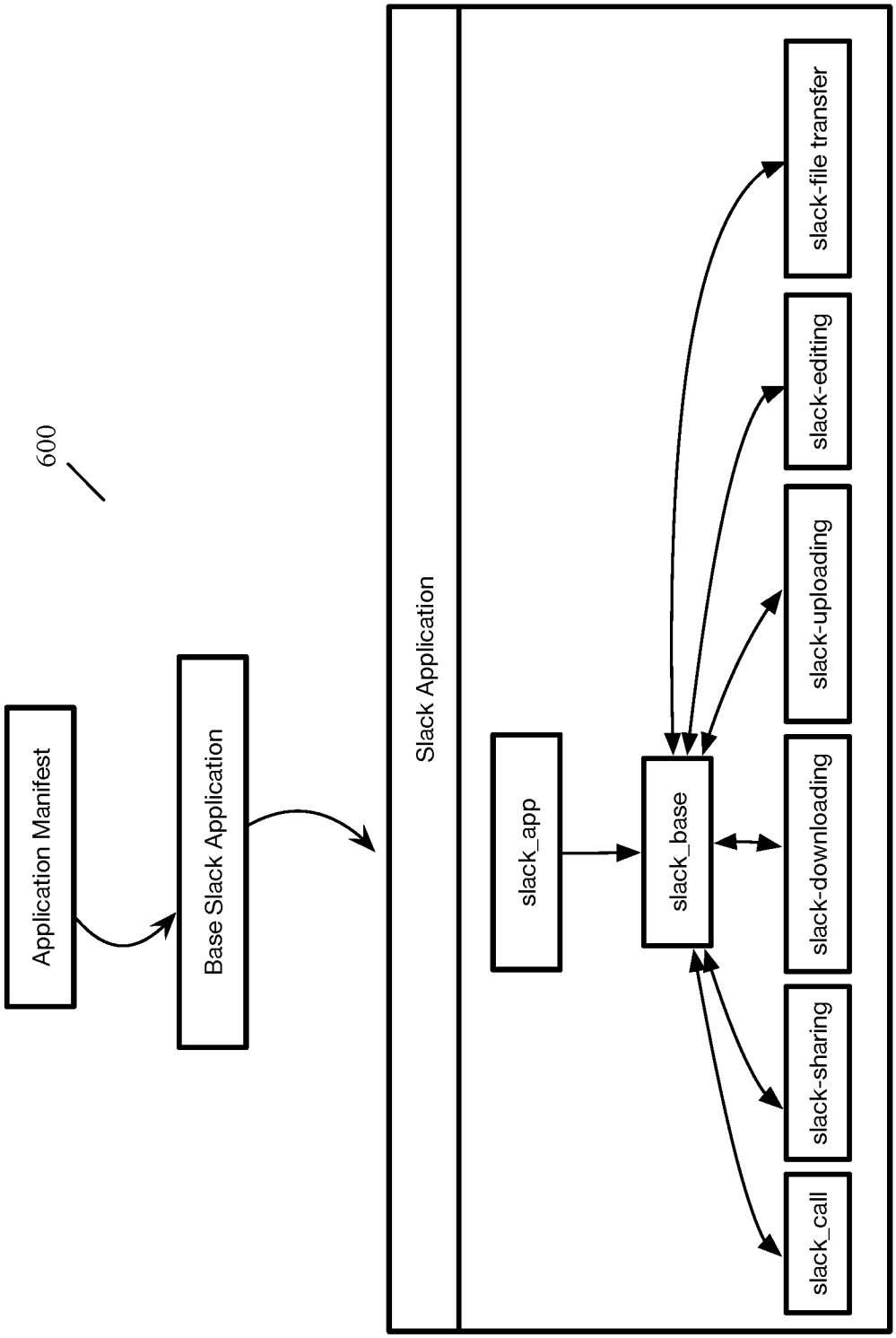

600
Application Manifest
Base Slack Application
Slack Application
slack_app
slack_base
slack_call
slack-sharing
slack-downloading
slack-uploading
slack-editing
slack-file transfer

Figure 6

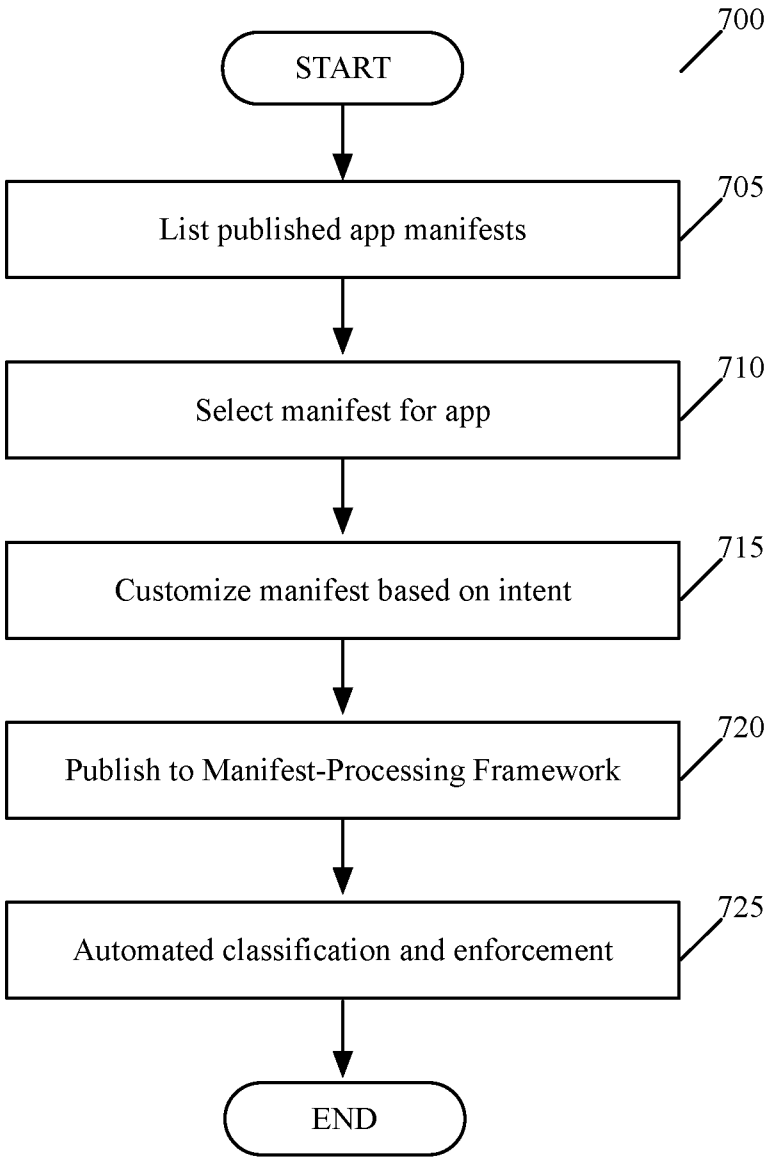

700
START
List published app manifests
705
Select manifest for app
710
Customize manifest based on intent
715
Publish to Manifest-Processing Framework
720
Automated classification and enforcement
725
END

Figure 7

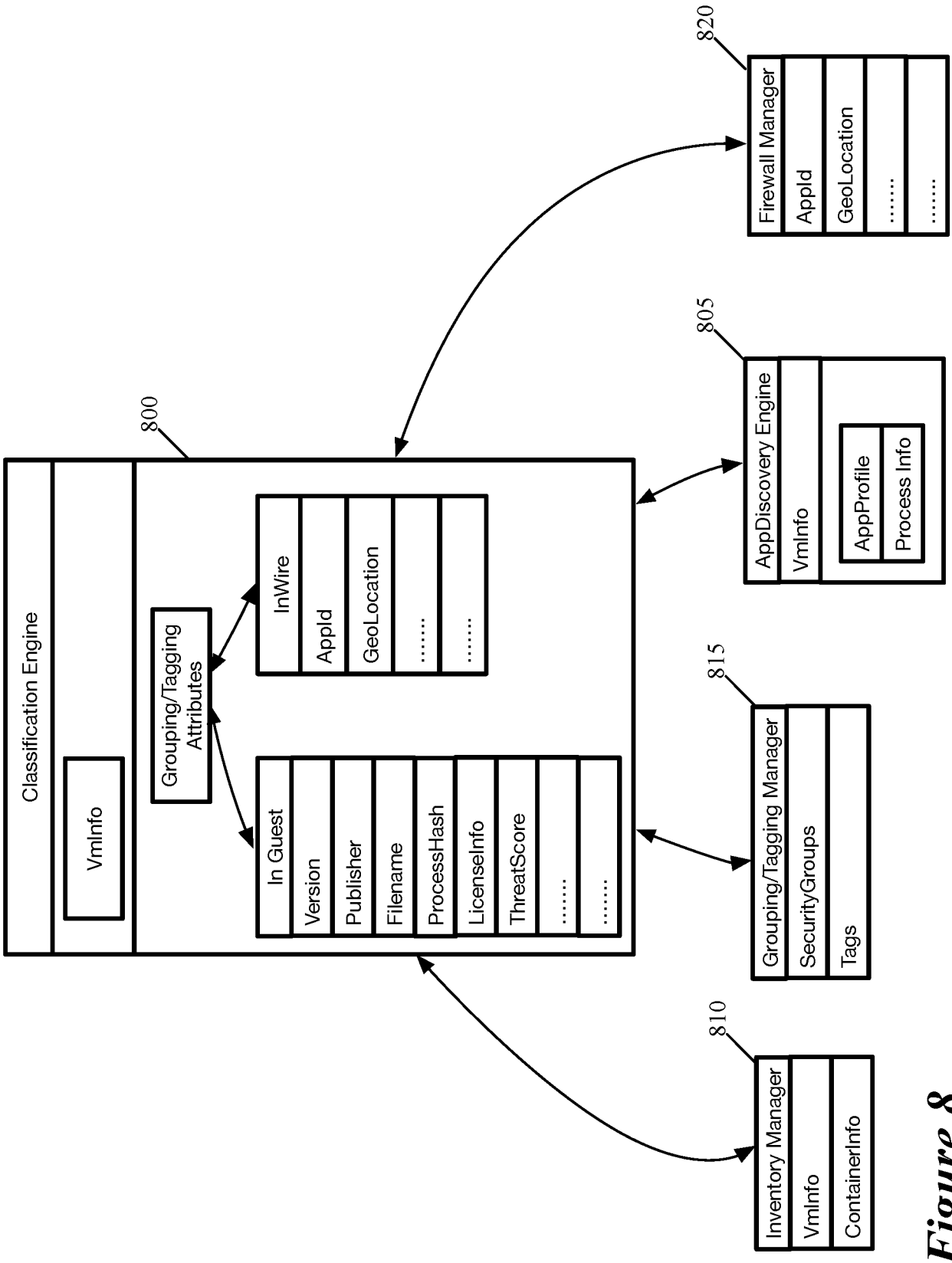

800
Classification Engine
VmInfo
Grouping/Tagging Attributes
InWire
AppId
GeoLocation
.....
.....
In Guest
Version
Publisher
Filename
ProcessHash
LicenseInfo
ThreatScore
.....
.....
820
Firewall Manager
AppId
GeoLocation
.....
.....
805
AppDiscovery Engine
VmInfo
AppProfile
Process Info
815
Grouping/Tagging Manager
SecurityGroups
Tags
810
Inventory Manager
VmInfo
ContainerInfo

Figure 8

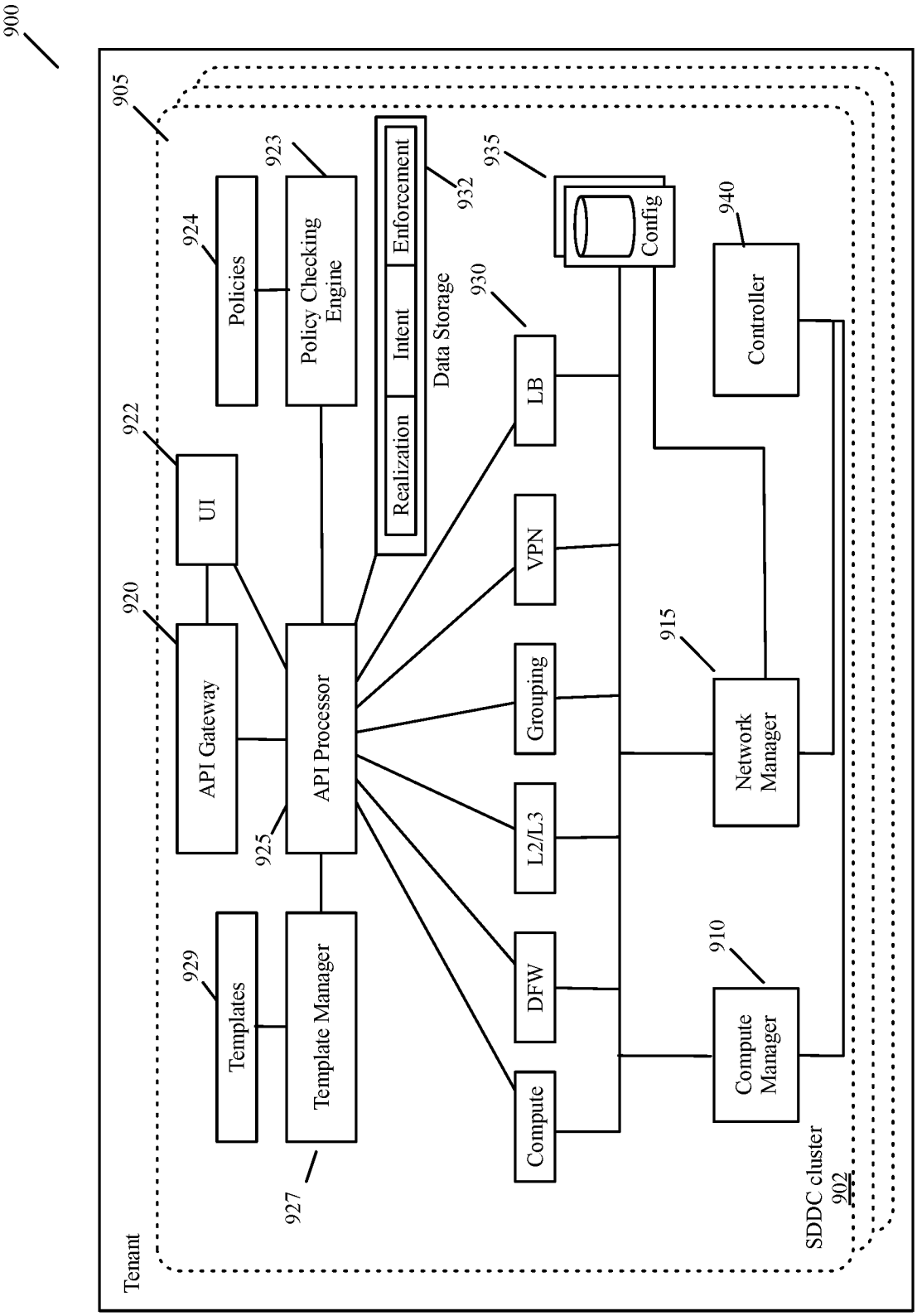

900
905
Tenant
SDDC cluster
902
923
924
Policies
Policy Checking Engine
922
UI
920
API Gateway
925
API Processor
927
929
Templates
Template Manager
Realization
Intent
Enforcement
Data Storage
932
935
Config
930
LB
VPN
Grouping
L2/L3
DFW
Compute
915
Network Manager
940
Controller
910
Compute Manager

Figure 9

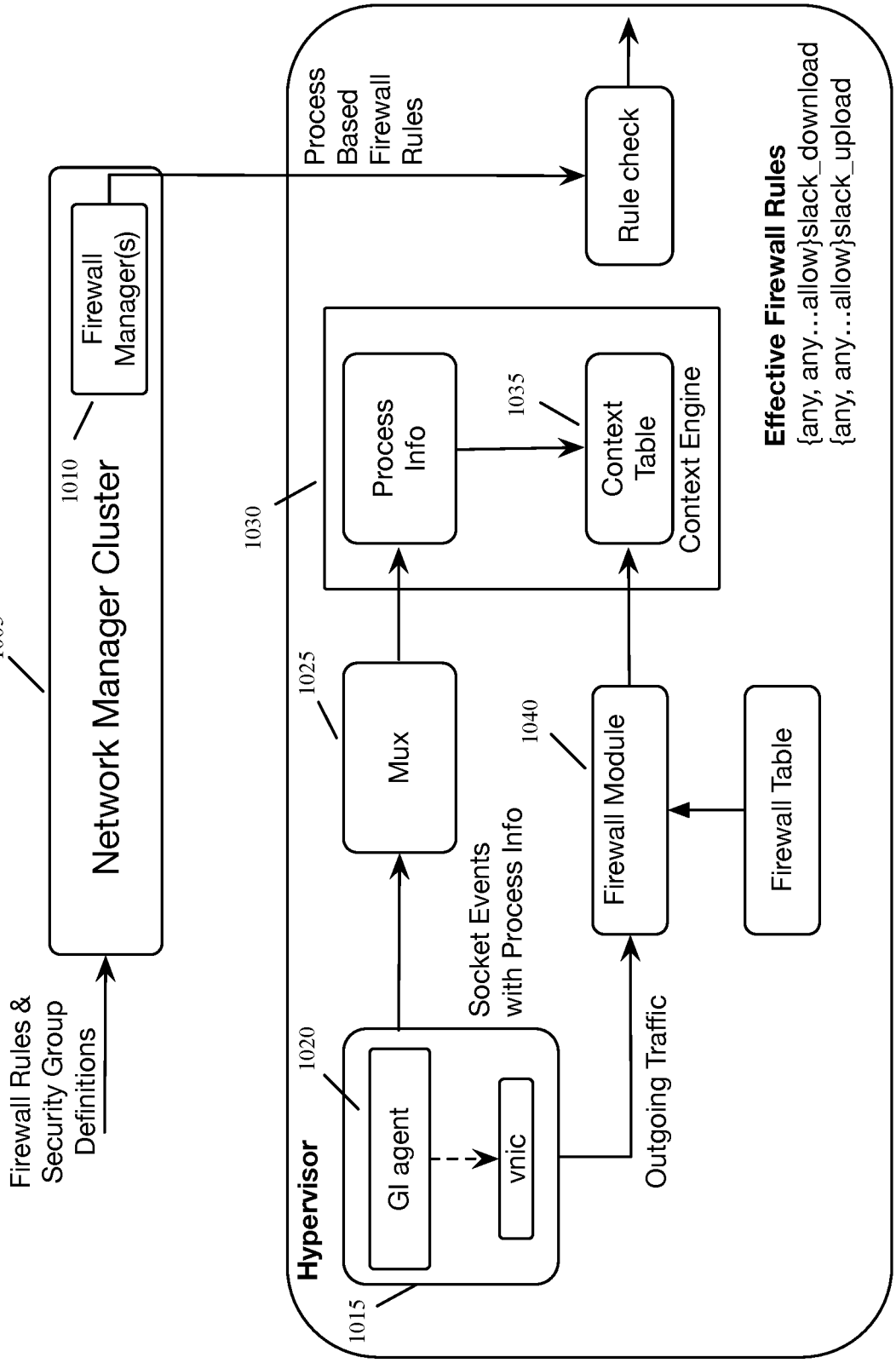

Firewall Rules & Security Group Definitions
1005
Network Manager Cluster
1010
Firewall Manager(s)
Hypervisor
1015
1020
GI agent
vnic
Socket Events with Process Info
1025
Mux
1030
Process Info
1035
Context Table
Context Engine
Process Based Firewall Rules
Rule check
Effective Firewall Rules
{any, any…allow}slack_download
{any, any…allow}slack_upload
Outgoing Traffic
1040
Firewall Module
Firewall Table

Figure 10

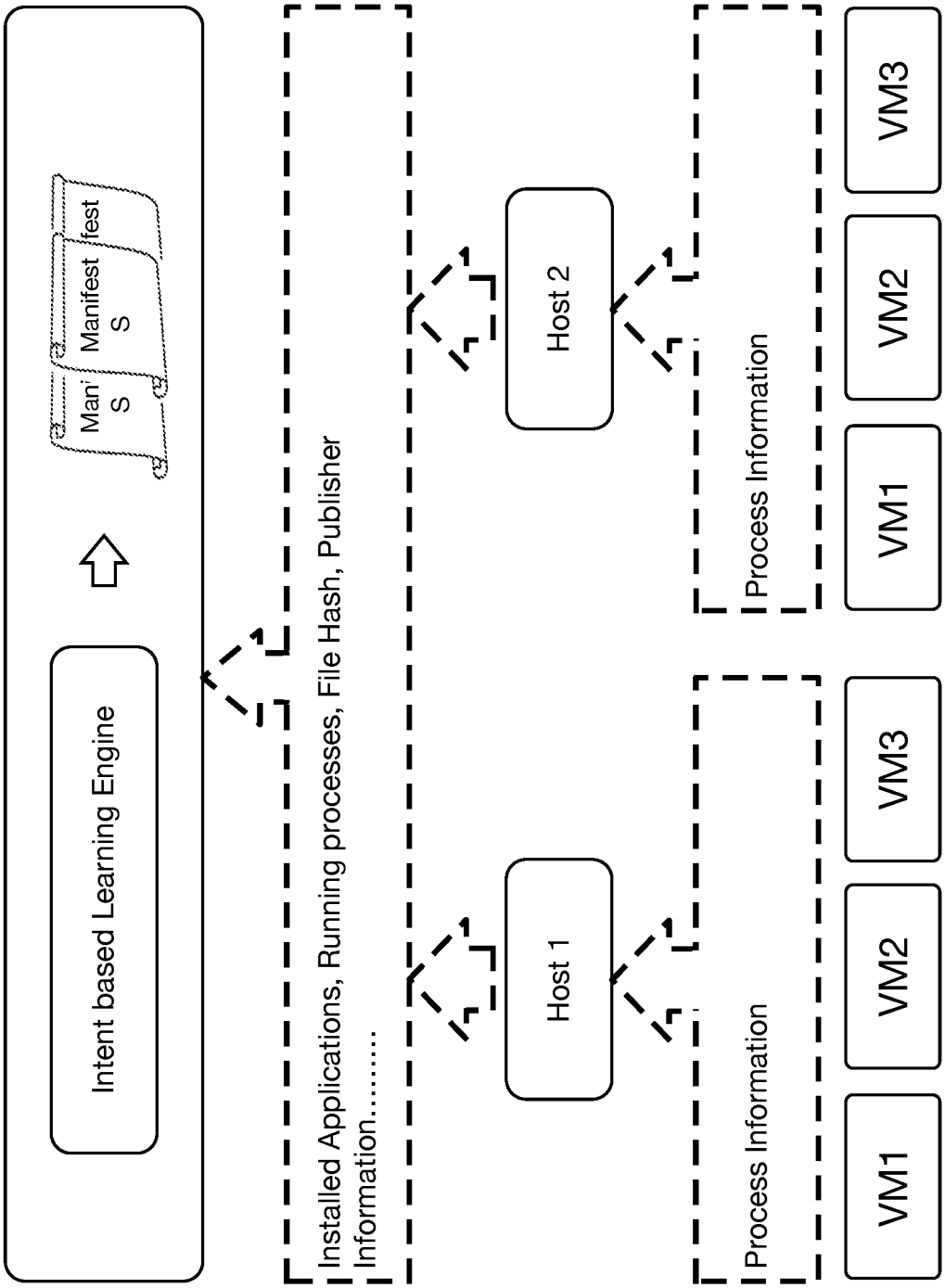

Intent based Learning Engine
Manifest
Manifest
Installed Applications, Running processes, File Hash, Publisher Information.....
Host 1
Host 2
Process Information
Process Information
VM1
VM2
VM3
VM1
VM2
VM3

Figure 11

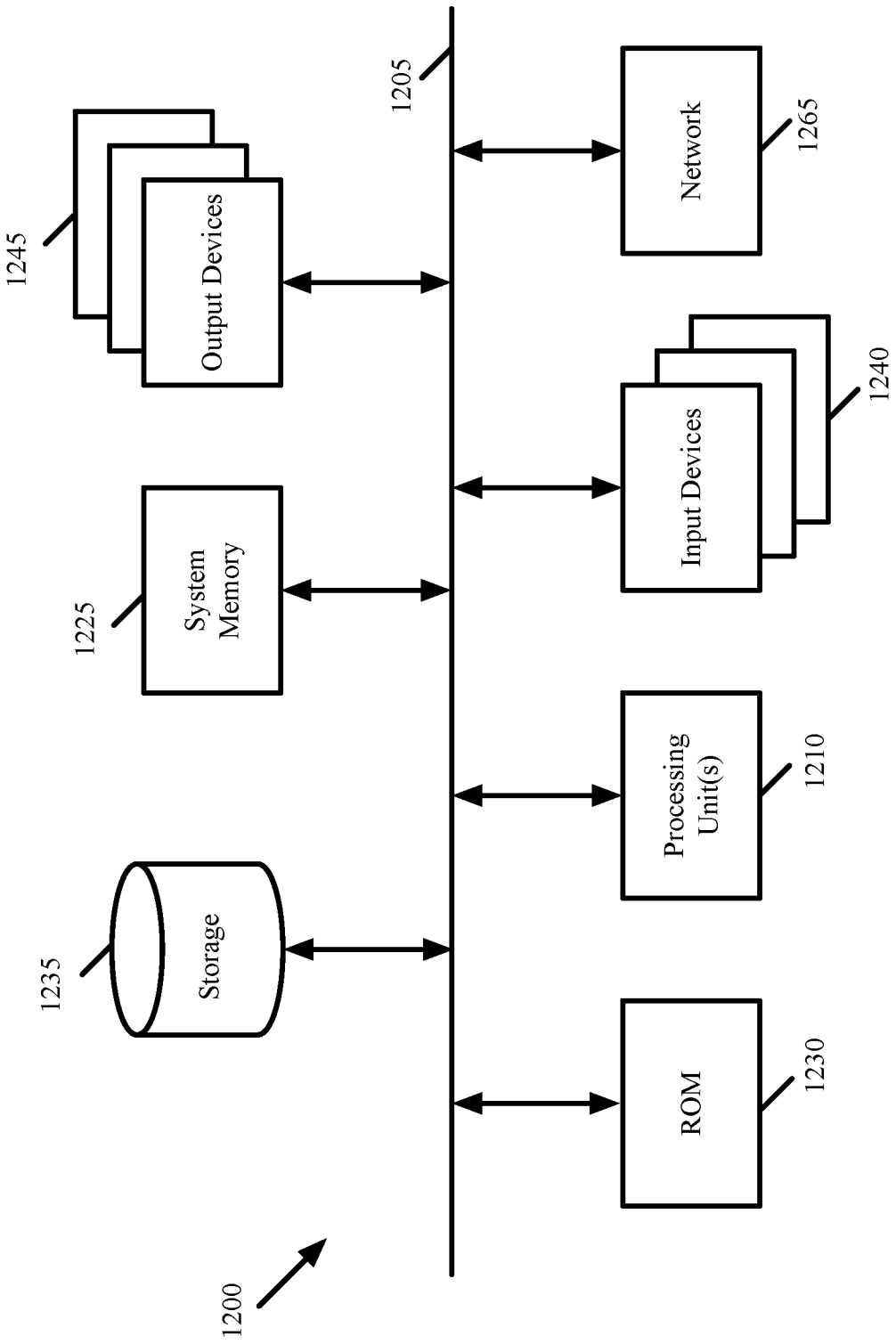

1200
1205
1210
Processing Unit(s)
1225
System Memory
1230
ROM
1235
Storage
1240
Input Devices
1245
Output Devices
1265
Network

Figure 12

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2019/046565

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/54 H04L29/06
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2018/183757 A1 (GUNDA LAXMIKANT [US] ET AL) 28 June 2018 (2018-06-28) abstract paragraph [0002] paragraph [0031] -----	1-29
X	US 2018/131675 A1 (SENGUPTA ANIRBAN [US] ET AL) 10 May 2018 (2018-05-10) paragraph [0019] - paragraph [0020] -----	1-29
X	US 2017/374106 A1 (HAMOU CLAUDE [US] ET AL) 28 December 2017 (2017-12-28) paragraph [0023] - paragraph [0027] -----	1-29
A	US 2013/019314 A1 (JI PENG [CN] ET AL) 17 January 2013 (2013-01-17) the whole document -----	1-29

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
18 October 2019	29/10/2019

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Dewyn, Torkild

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2019/046565

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2018183757 A1	28-06-2018	CN 110214311 A	06-09-2019
		EP 3542267 A1	25-09-2019
		US 2018183757 A1	28-06-2018
		WO 2018119164 A1	28-06-2018
US 2018131675 A1	10-05-2018	NONE	
US 2017374106 A1	28-12-2017	NONE	
US 2013019314 A1	17-01-2013	NONE	