Description

~~CROSS-REFERENCE TO OTHER APPLICATIONS~~

~~This application claims priority to U.S. Provisional Patent Application No. 62/590,986 entitled USING GRAPH-BASED MODELS TO IDENTIFY ANOMALIES filed Nov. 27, 2017, and also claims priority to U.S. Provisional Patent Application No. 62/650,971 entitled USING GRAPH-BASED MODELS TO IDENTIFY ANOMALIES filed Mar. 30, 2018, both of which applications are incorporated herein by reference for all purposes.~~

~~BACKGROUND OF THE INVENTION~~

~~Businesses and other entities often make use of data centers to provide a variety of computing resources. Increasingly, those resources are being virtualized. Such virtualization can provide benefits, such as efficient scalability and redundancy benefits. Unfortunately, such virtualization can also make it more difficult to detect and mitigate intruders (and other nefarious individuals).~~

BRIEF DESCRIPTION OF THE DRAWINGS

~~Various embodiments of the invention are disclosed in the following detailed description and the accompanying drawings.~~

~~FIG. 1~~The accompanying drawings illustrate various embodiments and are a part of the specification. The illustrated embodiments are merely examples and do not limit the scope of the disclosure. Throughout the drawings, identical or similar reference numbers designate identical or similar elements.

FIG. **1**A shows an illustrative configuration in which a data platform is configured to perform various operations with respect to a cloud environment that includes a plurality of compute assets.

FIG. **1**B shows an illustrative implementation of the configuration of FIG. **1**A.

FIG. **1**C illustrates an example computing device.

FIG. **1**D illustrates an example of an environment in which activities that occur within datacenters are modeled.

FIG.~~2~~ **2**A illustrates an example of a process, used by an agent, to collect and report information about a client.

FIG.~~3A illustrates an example of static information collected by an agent.~~

~~FIG. 3B illustrates an example of variable information collected by an agent.~~

~~FIG. 3C illustrates an example of histogram data.~~

~~FIG. 3D illustrates an example of histogram data.~~

~~FIG. 4~~ **2**B illustrates a 5-tuple of data collected by an agent, physically and logically.

FIG. **2**C~~FIG. 5~~ illustrates a portion of a polygraph.

FIG. **2**D~~FIG. 6~~ illustrates a portion of a polygraph.

FIG. **2**E~~FIG. 7~~ illustrates an example of a communication polygraph.

FIG. **2**F~~FIG. 8~~ illustrates an example of a polygraph.

FIG. **2**G~~FIG. 9~~ illustrates an example of a polygraph as rendered in an interface.

FIG. **2**H~~FIG. 10~~ illustrates an example of a portion of a polygraph as rendered in an interface.

FIG. **2**I~~FIG. 11~~ illustrates an example of a portion of a polygraph as rendered in an interface.

FIG. **2**J~~FIG. 12~~ illustrates an example of a portion of a polygraph as rendered in an interface.

FIG. **2**K~~FIG. 13~~ illustrates an example of a portion of a polygraph as rendered in an interface.

FIG. **2**L~~FIG. 14~~ illustrates an example of an insider behavior graph as rendered in an interface.

FIG. **2**M~~FIG. 15~~ illustrates an example of a privilege change graph as rendered in an interface.

FIG. 2NFIG. 16 illustrates an example of a user login graph as rendered in an interface.
FIG. 2OFIG. 17 illustrates an example of a machine server graph as rendered in an interface.
FIG.FIG. 18 3A illustrates an example of a process for detecting anomalies in a network environment.
FIG. 3BFIG. 19A depicts a set of example processes communicating with other processes.
FIG. 19BFIG. 3C depicts a set of example processes communicating with other processes.
FIG. 19CFIG. 3D depicts a set of example processes communicating with other processes.
FIG. 19DFIG. 3E depicts two pairs of clusters.
FIG. 20FIG. 3F is a representation of a user logging into a first machine, then into a second machine from the first machine, and then making an external connection.
FIG. 21 3G is an alternate representation of actions occurring in FIG. 20 3F.
FIG. 22 3H illustrates an example of a process for performing extended user tracking.
FIG. 23 3I is a representation of a user logging into a first machine, then into a second machine from the first machine, and then making an external connection.
FIG. 24 3J illustrates an example of a process for performing extended user tracking.
FIG. 25A 3K illustrates example records.
FIG. 25B 3L illustrates example output from performing an ssh connection match.
FIG. 25C 3M illustrates example records.
FIG. 25D 3N illustrates example records.
FIG. 25E 3O illustrates example records.
FIG. 25F 3P illustrates example records.
FIG. 25G 3Q illustrates an adjacency relationship between two login sessions.
FIG. 25H 3R illustrates example records.
FIG. 26 3S illustrates an example of a process for detecting anomalies.
FIG. 27A 4A illustrates a representation of an embodiment of an insider behavior graph.
FIG. 27B 4B illustrates an embodiment of a portion of an insider behavior graph.
FIG. 28A 4C illustrates an embodiment of a portion of an insider behavior graph.
FIG. 28B 4D illustrates an embodiment of a portion of an insider behavior graph.
FIG. 29 4E illustrates a representation of an embodiment of a user login graph.
FIG. 30 illustrates a representation of a process tree.
FIG. 31 4F illustrates an example of a privilege change graph.
FIG. 4GFIG. 32 illustrates an example of a privilege change graph.
FIG. 4HFIG. 33 illustrates an example of a user interacting with a portion of an interface.
FIG. 4IFIG. 34 illustrates an example of a dossier for an event.
FIG. 4JFIG. 35 illustrates an example of a dossier for a domain.
FIG. 36 illustrates an example of a card specification.
FIG. 37 illustrates an example of three user tracking data sources.
FIG. 38 depicts an example of card schema introspection.
FIG. 39 depicts an example of card schema introspection.
FIG. 40FIG. 4K depicts an example of an Entity Join Graphgraph by FilterKey and FilterKey Group (implicit join).
FIG. 4LFIG. 41 depicts an example introspection corresponding to the card depicted in FIG. 36.
FIG. 42 depicts an example query service log for a join path search.
FIG. 43 depicts an example of dynamically generated SQL.
FIG. 44 illustrates an example introspection corresponding to FIG. 37.
FIG. 45 depicts an example query service log for a join path search.

~~FIG. 46 depicts an example of dynamically generated SQL.~~
~~FIG. 47A depicts a filter request.~~
~~FIG. 47B depicts an SQL translation.~~
~~FIG. 47C depicts a join.~~
~~FIG. 47D depicts an SQL translation.~~
~~FIG. 48~~ illustrates an example of a process for dynamically generating and executing a query.
FIG. **5**A sets forth a system for providing many of the features described herein for user devices as a distributed edge service in accordance with some embodiments of the present disclosure.
FIG. **5**B sets forth a system for providing many of the features described herein for user devices as a distributed edge service in accordance with some embodiments of the present disclosure.
FIG. **6** sets forth a flow chart illustrating an example method of detecting deviations from typical user behavior in accordance with some embodiments of the present disclosure.
FIG. **7** sets forth a flow chart illustrating an additional example method of detecting deviations from typical behavior in accordance with some embodiments of the present disclosure.
FIG. **8** sets forth a flow chart illustrating an additional example method of detecting deviations from typical behavior in accordance with some embodiments of the present disclosure.
FIG. **9** sets forth a flow chart illustrating an example method of detecting a location of a user device in accordance with some embodiments of the present disclosure.
FIG. **10** sets forth a flow chart illustrating an additional example method of detecting a location of a user device in accordance with some embodiments of the present disclosure.
FIG. **11** sets forth a flow chart illustrating an additional example method of detecting a location of a user device in accordance with some embodiments of the present disclosure.
FIG. **12** sets forth a flow chart illustrating an example method of detecting deviation from normal behavior of a device in accordance with some embodiments of the present disclosure.
FIG.~~FIG. 49A illustrates a portion of a query builder library.~~
~~FIG. 49B illustrates three examples of SQL entities.~~
~~FIGS. 50A and 50B illustrate portions of an embodiment of a ProcessClusterFilters definition.~~
~~FIG. 51 illustrates an example of an introspection corresponding to a ProcessClusterFilters request.~~
~~FIG. 52A illustrates an example of a base table card.~~
~~FIG. 52B illustrates an example of a filter request.~~
~~FIG. 52C illustrates an example of a dynamically generated SQL query.~~
~~FIG. 52D illustrates an example of a filter request.~~
~~FIG. 52E illustrates an example of a dynamically generated SQL query.~~
~~FIG. 53A illustrates a card.~~
~~FIG. 53B illustrates a request.~~
~~FIG. 53C illustrates a dynamically generated SQL query.~~
~~FIG. 53D illustrates a request.~~
~~FIG. 53E illustrates a dynamically generated SQL query.~~
13 sets forth a flow chart illustrating an additional example method of detecting deviation from normal behavior of a device according to embodiments of the present disclosure.
FIG. **14** sets forth a flow chart illustrating an additional example method of detecting deviation from normal behavior of a device in some embodiments of the present disclosure.
FIG. **15**A sets forth an example of a user-specific polygraph in accordance with some embodiments of the present disclosure.

FIG. **15**B sets forth an example of a user-specific polygraph in accordance with some embodiments of the present disclosure.
FIG. **16** sets forth a flowchart of an example method of using activity monitored by multiple data sources to identify shadow systems according to some embodiments of the present disclosure.
FIG. **17** sets forth a flowchart of another example method of using activity monitored by multiple data sources to identify shadow systems according to some embodiments of the present disclosure.
FIG. **18** sets forth a flowchart of another example method of using activity monitored by multiple data sources to identify shadow systems according to some embodiments of the present disclosure.
FIG. **19** sets forth a flowchart of another example method of using activity monitored by multiple data sources to identify shadow systems according to some embodiments of the present disclosure.
FIG. **20** sets forth an example polygraph for identifying shadow systems according to some embodiments of the present disclosure.

DETAILED DESCRIPTION

~~The invention~~Various illustrative embodiments are described herein with reference to the accompanying drawings. It will, however, be evident that various modifications and changes may be made thereto, and additional embodiments may be implemented, without departing from the scope of the invention as set forth in the claims. For example, certain features of one embodiment described herein may be combined with or substituted for features of another embodiment described herein. The description and drawings are accordingly to be regarded in an illustrative rather than a restrictive sense.

FIG. **1**A shows an illustrative configuration **10** in which a data platform **12** is configured to perform various operations with respect to a cloud environment **14** that includes a plurality of compute assets **16**-**1** through **16**-N (collectively "compute assets **16**). For example, data platform **12** may include data ingestion resources **18** configured to ingest data from cloud environment **14** into data platform **12**, data processing resources **20** configured to perform data processing operations with respect to the data, user interface resources **22** configured to provide one or more external users and/or compute resources (e.g., computing device **24**) with access to an output of data processing resources **20**. Each of these resources are described in detail herein.

Cloud environment **14** may include any suitable network-based computing environment as may serve a particular application. For example, cloud environment **14** may be implemented by one or more compute resources provided and/or otherwise managed by one or more cloud service providers, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and/or any other cloud service provider configured to provide public and/or private access to network-based compute resources.

Compute assets **16** may include, but are not limited to, containers (e.g., container images, deployed and executing container instances, etc.), virtual machines, workloads, applications, processes, physical machines, compute nodes, clusters of compute nodes, software runtime environments (e.g., container runtime environments), and/or any other virtual and/or physical compute resource that may reside in and/or be executed by one or more computer resources in cloud environment **14**. In some examples, one or more compute assets **16** may reside in one or more datacenters.

A compute asset **16** may be associated with (e.g., owned, deployed, or managed by) a particular entity, such as a customer or client of cloud environment **14** and/or data platform **12**.

Accordingly, for purposes of the discussion herein, cloud environment **14** may be used by one or more entities.

Data platform **12** may be configured to perform one or more data security monitoring and/or remediation services, compliance monitoring services, anomaly detection services, DevOps services, compute asset management services, and/or any other type of data analytics service as may serve a particular implementation. Data platform **12** may be managed or otherwise associated with any suitable data platform provider, such as a provider of any of the data analytics services described herein. The various resources included in data platform **12** may reside in the cloud and/or be located on-premises and be implemented by any suitable combination of physical and/or virtual compute resources, such as one or more computing devices, microservices, applications, etc.

Data ingestion resources **18** may be configured to ingest data from cloud environment **14** into data platform **12**. This may be performed in various ways, some of which are described in detail herein. For example, as illustrated by arrow **26**, data ingestion resources **18** may be configured to receive the data from one or more agents deployed within cloud environment **14**, utilize an event streaming platform (e.g., Kafka) to obtain the data, and/or pull data (e.g., configuration data) from cloud environment **14**. In some examples, data ingestion resources **18** may obtain the data using one or more agentless configurations.

The data ingested by data ingestion resources **18** from cloud environment **14** may include any type of data as may serve a particular implementation. For example, the data may include data representative of configuration information associated with compute assets **16**, information about one or more processes running on compute assets **16**, network activity information, information about events (creation events, modification events, communication events, user-initiated events, etc.) that occur with respect to compute assets **16**, etc. In some examples, the data may or may not include actual customer data processed or otherwise generated by compute assets **16**.

As illustrated by arrow **28**, data ingestion resources **18** may be configured to load the data ingested from cloud environment **14** into a data store **30**. Data store **30** is illustrated in FIG. **1**A as being separate from and communicatively coupled to data platform **12**. However, in some alternative embodiments, data store **30** is included within data platform **12**.

Data store **30** may be implemented by any suitable data warehouse, data lake, data mart, and/or other type of database structure as may serve a particular implementation. Such data stores may be proprietary or may be embodied as vendor provided products or services such as, for example, Snowflake, Google BigQuery, Druid, Amazon Redshift, IBM Db2, Dremio, Databricks Lakehouse Platform, Cloudera, Azure Synapse Analytics, and others.

Although the examples described herein largely relate to embodiments where data is collected from agents and ultimately stored in a data store such as those provided by Snowflake, in other embodiments data that is collected from agents and other sources may be stored in different ways. For example, data that is collected from agents and other sources may be stored in a data warehouse, data lake, data mart, and/or any other data store.

A data warehouse may be embodied as an analytic database (e.g., a relational database) that is created from two or more data sources. Such a data warehouse may be leveraged to store historical data, often on the scale of petabytes. Data warehouses may have compute and memory resources for running complicated queries and generating reports. Data warehouses may be the data sources for business intelligence ('BI') systems, machine learning applications, and/or other applications. By leveraging a data warehouse, data that has been copied into the data warehouse may be indexed for good analytic query performance, without affecting the write performance of

a database (e.g., an Online Transaction Processing ('OLTP') database). Data warehouses also enable the joining data from multiple sources for analysis. For example, a sales OLTP application probably has no need to know about the weather at various sales locations, but sales predictions could take advantage of that data. By adding historical weather data to a data warehouse, it would be possible to factor it into models of historical sales data.

Data lakes, which store files of data in their native format, may be considered as "schema on read" resources. As such, any application that reads data from the lake may impose its own types and relationships on the data. Data warehouses, on the other hand, are "schema on write," meaning that data types, indexes, and relationships are imposed on the data as it is stored in an enterprise data warehouse (EDW). "Schema on read" resources may be beneficial for data that may be used in several contexts and poses little risk of losing data. "Schema on write" resources may be beneficial for data that has a specific purpose, and good for data that must relate properly to data from other sources. Such data stores may include data that is encrypted using homomorphic encryption, data encrypted using privacy-preserving encryption, smart contracts, non-fungible tokens, decentralized finance, and other techniques.

Data marts may contain data oriented towards a specific business line whereas data warehouses contain enterprise-wide data. Data marts may be dependent on a data warehouse, independent of the data warehouse (e.g., drawn from an operational database or external source), or a hybrid of the two. In embodiments described herein, different types of data stores (including combinations thereof) may be leveraged.

Data processing resources **20** may be configured to perform various data processing operations with respect to data ingested by data ingestion resources **18**, including data ingested and stored in data store **30**. For example, data processing resources **20** may be configured to perform one or more data security monitoring and/or remediation operations, compliance monitoring operations, anomaly detection operations, DevOps operations, compute asset management operations, and/or any other type of data analytics operation as may serve a particular implementation. Various examples of operations performed by data processing resources **20** are described herein.

As illustrated by arrow **32**, data processing resources **20** may be configured to access data in data store **30** to perform the various operations described herein. In some examples, this may include performing one or more queries with respect to the data stored in data store **30**. Such queries may be generated using any suitable query language.

In some examples, the queries provided by data processing resources **20** may be configured to direct data store **30** to perform one or more data analytics operations with respect to the data stored within data store **30**. These data analytics operations may be with respect to data specific to a particular entity (e.g., data residing in one or more silos within data store **30** that are associated with a particular customer) and/or data associated with multiple entities. For example, data processing resources **20** may be configured to analyze data associated with a first entity and use the results of the analysis to perform one or more operations with respect to a second entity.

One or more operations performed by data processing resources **20** may be performed periodically according to a predetermined schedule. For example, one or more operations may be performed by processing resources **20** every hour or any other suitable time interval.

Additionally or alternatively, one or more operations performed by data processing resources **20** may be performed in substantially real-time (or near real-time) as data is ingested into data platform **12**. In this manner, the results of such operations (e.g., one or more detected

anomalies in the data) may be provided to one or more external entities (e.g., computing device **24** and/or one or more users) in substantially real-time and/or in near real-time.
User interface resources **22** may be configured to perform one or more user interface operations, examples of which are described herein. For example, user interface resources **22** may be configured to present one or more results of the data processing performed by data processing resources **20** to one or more external entities (e.g., computing device **24** and/or one or more users), as illustrated by arrow **34**. As illustrated by arrow **36**, user interface resources **22** may access data in data store **30** to perform the one or more user interface operations
FIG. **1**B illustrates an implementation of configuration **10** in which an agent **38** (e.g., agent **38-1** through agent **38**-N) is installed on each of compute assets **16**. As used herein, an agent may include a self-contained binary and/or other type of code or application that can be run on any appropriate platforms, including within containers and/or other virtual compute assets. Agents **38** may monitor the nodes on which they execute for a variety of different activities, including but not limited to, connection, process, user, machine, and file activities. In some examples, agents **38** can be executed in user space, and can use a variety of kernel modules (e.g., auditd, iptables, netfilter, pcap, etc.) to collect data. Agents can be implemented in any appropriate programming language, such as C or Golang, using applicable kernel APIs.
Agents **38** may be deployed in any suitable manner. For example, an agent **38** may be deployed as a containerized application or as part of a containerized application. As described herein, agents **38** may selectively report information to data platform **12** in varying amounts of detail and/or with variable frequency.
Also shown in FIG. **1**B is a load balancer **40** configured to perform one or more load balancing operations with respect to data ingestion operations performed by data ingestion resources **18** and/or user interface operations performed by user interface resources **22**. Load balancer **40** is shown to be included in data platform **12**. However, load balancer **40** may alternatively be located external to data platform **12**. Load balancer **40** may be implemented by any suitable microservice, application, and/or other computing resources. In some alternative examples, data platform **12** may not utilize a load balancer such as load balancer **40**.
Also shown in FIG. **1**B is long term storage **42** with which data ingestion resources may interface, as illustrated by arrow **44**. Long term storage **42** may be implemented by any suitable type of storage resources, such as cloud-based storage (e.g., AWS S3, etc.) and/or on-premises storage and may be used by data ingestion resources **18** as part of the data ingestion process. Examples of this are described herein. In some examples, data platform **12** may not utilize long term storage **42**.
The embodiments described herein can be implemented in numerous ways, including as a process; an apparatus; a system; a composition of matter; a computer program product embodied on a computer readable storage medium; and/or a processor, such as a processor configured to execute instructions stored on and/or provided by a memory coupled to the processor. In this specification, these implementations, or any other form that the invention may take, may be referred to as techniques. In general, the order of the steps of disclosed processes may be altered within the scope of the ~~invention.~~principles described herein. Unless stated otherwise, a component such as a processor or a memory described as being configured to perform a task may be implemented as a general component that is temporarily configured to perform the task at a given time or a specific component that is manufactured to perform the task. As used herein, the term 'processor' refers to one or more devices, circuits, and/or processing cores configured to process data, such as computer program instructions.

~~A detailed description of one or more embodiments of the invention is provided below along with accompanying figures that illustrate the principles of the invention. The invention is described in connection with such embodiments, but the invention is not limited to any embodiment. The scope of the invention is limited only by the claims and the invention encompasses numerous alternatives, modifications and equivalents. Numerous specific details are set forth in the following description in order to provide a thorough understanding of the invention. These details are provided for the purpose of example and the invention may be practiced according to the claims without some or all of these specific details. For the purpose of clarity, technical material that is known in the technical fields related to the invention has not been described in detail so that the invention is not unnecessarily obscured.~~

~~A detailed description of one or more embodiments of the invention is provided below along with accompanying figures that illustrate the principles of the invention. The invention is described in connection with such embodiments, but the invention is not limited to any embodiment. The scope of the invention is limited only by the claims and the invention encompasses numerous alternatives, modifications and equivalents. Numerous specific details are set forth in the following description in order to provide a thorough understanding of the invention. These details are provided for the purpose of example and the invention may be practiced according to the claims without some or all of these specific details. For the purpose of clarity, technical material that is known in the technical fields related to the invention has not been described in detail so that the invention is not unnecessarily obscured.~~

In some examples, a non-transitory computer-readable medium storing computer-readable instructions may be provided in accordance with the principles described herein. The instructions, when executed by a processor of a computing device, may direct the processor and/or computing device to perform one or more operations, including one or more of the operations described herein. Such instructions may be stored and/or transmitted using any of a variety of known computer-readable media.

A non-transitory computer-readable medium as referred to herein may include any non-transitory storage medium that participates in providing data (e.g., instructions) that may be read and/or executed by a computing device (e.g., by a processor of a computing device). For example, a non-transitory computer-readable medium may include, but is not limited to, any combination of non-volatile storage media and/or volatile storage media. Exemplary non-volatile storage media include, but are not limited to, read-only memory, flash memory, a solid-state drive, a magnetic storage device (e.g. a hard disk, a floppy disk, magnetic tape, etc.), ferroelectric random-access memory ("RAM"), and an optical disc (e.g., a compact disc, a digital video disc, a Blu-ray disc, etc.). Exemplary volatile storage media include, but are not limited to, RAM (e.g., dynamic RAM).

FIG. ~~1~~ **1**C illustrates an- example computing device **50** that may be specifically configured to perform one or more of the processes described herein. Any of the systems, microservices, computing devices, and/or other components described herein may be implemented by computing device **50**.

As shown in FIG. **1**C, computing device **50** may include a communication interface **52**, a processor **54**, a storage device **56**, and an input/output ("I/O") module **58** communicatively connected one to another via a communication infrastructure **60**. While an exemplary computing device **50** is shown in FIG. **1**C, the components illustrated in FIG. **1**C are not intended to be limiting. Additional or alternative components may be used in other embodiments. Components of computing device **50** shown in FIG. **1**C will now be described in additional detail.

Communication interface **52** may be configured to communicate with one or more computing devices. Examples of communication interface **52** include, without limitation, a wired network interface (such as a network interface card), a wireless network interface (such as a wireless network interface card), a modem, an audio/video connection, and any other suitable interface.
Processor **54** generally represents any type or form of processing unit capable of processing data and/or interpreting, executing, and/or directing execution of one or more of the instructions, processes, and/or operations described herein. Processor **54** may perform operations by executing computer-executable instructions **62** (e.g., an application, software, code, and/or other executable data instance) stored in storage device **56**.
Storage device **56** may include one or more data storage media, devices, or configurations and may employ any type, form, and combination of data storage media and/or device. For example ~~of~~, storage device **56** may include, but is not limited to, any combination of the non-volatile media and/or volatile media described herein. Electronic data, including data described herein, may be temporarily and/or permanently stored in storage device **56**. For example, data representative of computer-executable instructions **62** configured to direct processor **54** to perform any of the operations described herein may be stored within storage device **56**. In some examples, data may be arranged in one or more databases residing within storage device **56**.
I/O module **58** may include one or more I/O modules configured to receive user input and provide user output. I/O module **58** may include any hardware, firmware, software, or combination thereof supportive of input and output capabilities. For example, I/O module **58** may include hardware and/or software for capturing user input, including, but not limited to, a keyboard or keypad, a touchscreen component (e.g., touchscreen display), a receiver (e.g., an RF or infrared receiver), motion sensors, and/or one or more input buttons.
I/O module **58** may include one or more devices for presenting output to a user, including, but not limited to, a graphics engine, a display (e.g., a display screen), one or more output drivers (e.g., display drivers), one or more audio speakers, and one or more audio drivers. In certain embodiments, I/O module **58** is configured to provide graphical data to a display for presentation to a user. The graphical data may be representative of one or more graphical user interfaces and/or any other graphical content as may serve a particular implementation.
FIG. **1**D illustrates an example implementation **100** of configuration **10**. As such, one more components shown in FIG. **1**D may implement one or more components shown in FIG. **1**A and/or FIG. **1**B. In particular, implementation **100** illustrates an environment in which activities that occur within datacenters are modeled using data platform **12**. Using techniques described herein, a baseline of datacenter activity can be modeled, and deviations from that baseline can be identified as anomalous. Anomaly detection can be beneficial in a security context, ~~and can also be useful in other contexts, such as devops~~a compliance context, an asset management context, a DevOps context, and/or any other data analytics context as may serve a particular implementation.
Two example datacenters (**104** and **106**) are shown in FIG. ~~1~~ 1D, and are associated with (e.g., belong to ~~fictional companies ACME~~) entities named entity A and ~~BETA~~entity B, respectively. A datacenter ~~can comprise~~may include dedicated equipment (e.g., owned and operated by ~~ACME~~entity A, or owned/leased by ~~ACME~~entity A and operated exclusively on ~~ACME's~~entity A's behalf by a third party). A datacenter can also ~~comprise~~include cloud-based resources, such as infrastructure as a service (IaaS), platform as a service (PaaS), and/or software as a service (SaaS) elements. The techniques described herein can be used in conjunction with multiple types

of datacenters, including ones wholly using dedicated equipment, ones that are entirely cloud-based, and ones that use a mixture of both dedicated equipment and cloud-based resources. Both datacenter **104** and datacenter **106** include a plurality of nodes, depicted collectively as set of nodes **108** and set of nodes **110**, respectively, in FIG. ~~1~~ 1D. These nodes may implement compute assets **16**. Installed on each of the nodes are in-server/in-virtual machine (VM)/embedded in IoT device agents (~~as applicable~~e.g., agent **112**), which are configured to collect data and report it to data platform ~~102~~**12** for analysis. ~~Agents are~~As described herein, agents may be small, self-contained binaries that can be run on any appropriate platforms, including virtualized ones (and, as applicable, within containers). Agents may monitor the nodes on which they execute for a variety of different activities, including: connection, process, user, machine, and file activities. Agents can be executed in user space, and can use a variety of kernel modules (e.g., auditd, iptables, netfilter, pcap, etc.) to collect data. Agents can be implemented in any appropriate programming language, such as C or Golang, using applicable kernel APIs.

As ~~will be~~ described ~~in more detail below~~herein, agents can selectively report information to data platform ~~102~~**12** in varying amounts of detail and/or with variable frequency. As ~~will~~is also ~~be~~ described ~~in more detail below~~herein, the data collected by agents ~~is~~may be used by data platform ~~102~~**12** to create polygraphs, which are graphs of logical entities, connected by behaviors. In some embodiments, agents report information directly to data platform ~~102~~**12**. In other embodiments, at least some agents provide information to a data aggregator, such as data aggregator **114**, which in turn provides information to data platform ~~102~~**12**. The functionality of a data aggregator can be implemented as a separate binary or other application (distinct from an agent binary), and can also be implemented by having an agent execute in an "aggregator mode" in which the designated aggregator node acts as a Layer **7** proxy for other agents that do not have access to data platform ~~102~~**12**. Further, a chain of multiple aggregators can be used, if applicable (e.g., with agent **112** providing data to data aggregator **114**, which in turn provides data to another aggregator (not pictured) which provides data to data platform ~~102~~**12**). An example way to implement an aggregator is through a program written in an appropriate language, such as C or Golang.

Use of an aggregator can be beneficial in sensitive environments (e.g., involving financial or medical transactions) where various nodes are subject to regulatory or other architectural requirements (e.g., prohibiting a given node from communicating with systems outside of datacenter **104**). ~~And, use~~Use of an aggregator can also help to minimize security exposure more generally. As one example, by limiting communications with data platform ~~102~~**12** to data aggregator **114**, individual nodes in ~~group~~nodes **108** need not make external network connections (e.g., via Internet **124**), which can potentially expose them to compromise (e.g., by other external devices, such as device **118**, operated by a criminal). Similarly, data platform ~~102~~**12** can provide updates, configuration information, etc., to data aggregator **114** (which in turn distributes them to nodes **108**), rather than requiring nodes **108** to allow incoming connections from data platform ~~102~~**12** directly.

Another benefit of an aggregator model is that network congestion can be reduced (e.g., with a single connection being made at any given time between data aggregator **114** and data platform ~~102~~**12**, rather than potentially many different connections being open between various of nodes **108** and data platform ~~102~~**12**). Similarly, network consumption can also be reduced (e.g., with the aggregator applying compression techniques/bundling data received from multiple agents).

One example way that an agent (e.g., agent **112**, installed on node **116**) can provide information to- data aggregator **114** is via a REST API, formatted using data serialization protocols such as Apache Avro. One example type of information sent by agent **112** to- data aggregator **114** is status information. Status information ~~is~~may be sent by an agent periodically (e.g., once an hour~~)~~ ~~and includes (~~ or once any other predetermined amount of time). Alternatively, status information may be sent continuously or in ~~various embodiments):~~response to occurrence of one or more events. The status information may include, but is not limited to, a. an amount of event backlog (in bytes) that has not yet been transmitted, b. configuration information, c. any data loss period for which data was dropped, d. a cumulative count of errors encountered since the agent started, e. version information for the agent binary, and/or f. cumulative statistics on data collection (e.g., number of network packets processed, new processes seen, etc.).

A second example type of information that may be sent by agent **112** to- data aggregator **114** is event data (described in more detail ~~below~~herein), which ~~includes~~may include a UTC timestamp for each event. As applicable, the agent can control the amount of data that it sends to the data aggregator in each call (e.g., a maximum of 10 MB) by adjusting the amount of data sent to manage the conflicting goals of transmitting data as soon as possible, and maximizing throughput. Data can also be compressed or uncompressed by the agent (as applicable) prior to sending the data.

Each data aggregator ~~runs~~may run within a particular customer environment. A data aggregator (e.g., data aggregator **114**) ~~facilitates~~may facilitate data routing from many different agents (e.g., agents executing on nodes **108**) to data platform ~~102~~**12**. In various embodiments,- data aggregator **114** ~~implements~~may implement a SOCKS **5** caching proxy through which agents can connect to data platform ~~102~~**12**. As applicable,- data aggregator **114** can encrypt (or otherwise obfuscate) sensitive information prior to transmitting it to data platform ~~102~~**12**, and can also distribute key material to agents which can encrypt the information (as applicable).- Data aggregator **114** ~~includes~~may include a local storage, to which agents can upload data (e.g., pcap packets). The storage ~~has~~may have a key-value interface. The local storage can also be omitted, and agents configured to upload data to a cloud storage or other storage area, as applicable.- Data aggregator **114** can, in some embodiments, also cache locally and distribute software upgrades, patches, or configuration information (e.g., as received from data platform ~~102~~**12**).

~~I. Agent Data Collection and Reporting~~

~~A. Initial Configuration~~

Various examples associated with agent data collection and reporting will now be described.

In the following example, suppose that a user (e.g., a network administrator) at ~~ACME~~entity A (hereinafter "~~Alice~~user A") has decided to begin using the services of data platform ~~102~~**12**. In some embodiments, ~~Alice accesses~~user A may access a web frontend (e.g., web app **120**) using ~~her~~ a computer ~~(~~ **126**~~)~~ and enrolls (on behalf of ~~ACME~~entity A) an account with data platform ~~102~~**12**. After enrollment is complete, ~~Alice is~~user A may be presented with a set of installers, pre-built and customized for the ~~ACME~~ environment of entity A, that ~~she~~user A can download from data platform ~~102~~**12** and deploy on nodes **108**. Examples of such installers include~~:~~, but are not limited to, a Windows executable file~~:~~, an iOS app~~; and~~, a Linux package (e.g., .deb or .rpm), a binary, or a container (e.g., a Docker container~~)~~. When a user (e.g., a network administrator) at ~~BETA~~entity B (hereinafter "~~Bob~~user B") also signs up for the services of data platform ~~102, he will~~ **12**, user B may be similarly ~~be~~ presented with a set of installers that are pre-built and customized for the ~~BETA~~ environment of entity B.

~~Alice~~User A deploys an appropriate installer on each of nodes **108** (e.g., with a Windows executable file deployed on a Windows-based platform~~, and~~ or a Linux package deployed on a Linux platform, as applicable). As applicable, the agent can be deployed in a container. Agent deployment can also be performed using one or more appropriate automation tools, such as Chef, Puppet, Salt, and Ansible. Deployment can also be performed using managed/hosted container management/orchestration frameworks such as Kubernetes, ~~mesos~~Mesos, and/or Docker ~~swarm~~Swarm.

In various embodiments, the agent ~~is~~may be installed in the user space (i.e., is not a kernel module), and the same binary is executed on each node of the same type (e.g., all Windows-based platforms have the same Windows-based binary installed on them). ~~The primary job~~An illustrative function of an agent, such as agent **112**, is to collect data (e.g., associated with node **116**) and report it (e.g., to data aggregator **114**). Other tasks that can be performed by agents include data configuration and upgrading.

~~B. Data Collection~~

One approach to collecting data as described herein is to collect virtually all information available about a node (and, e.g., the processes running on it). ~~A downside of this approach is that collecting information can be resource intensive (i.e., requiring potentially large amounts of processor, network, and storage resources). Another downside of this approach is that it can result in the collection of private or other sensitive information that can be difficult to filter/exclude from downstream reporting. An alternate approach is for~~Alternatively, the agent ~~to~~may monitor for network connections, and then begin collecting information about ~~those~~ processes associated with the network connections, using the presence of a network packet associated with a process as a trigger for collecting additional information about the process. As an example, if a user of ~~-~~ node **116** executes an application, such as a calculator application, which does not typically interact with the network, no information about use of that application ~~is~~may be collected by agent **112**~~/~~ and/or sent to ~~-~~ data aggregator **114**. If, however, the user of ~~-~~ node **116** executes an ssh command (e.g., to ssh from ~~-~~ node **116** to node **122**), agent **112** ~~will~~may collect information about the process and provide associated information to ~~-~~ data aggregator **114**. In various embodiments, the agent may always ~~collects/reports~~collect/report information about certain events, such as privilege escalation, irrespective of whether the event is associated with network activity.

An approach to collecting information (e.g., by an agent) is as follows, and described in conjunction with process **200** depicted in FIG. ~~2~~ 2A. An agent (e.g., agent **112**) monitors its node (e.g., node **116**) for network activity. One example way that agent **112** can monitor ~~-~~ node **116** for network activity is by using a network packet capture tool (e.g., listening using libpcap). As packets are received (~~202~~201), the agent obtains and maintains (e.g., in an in ~~-~~memory cache) connection information associated with the network activity (~~204~~202). Examples of such information include DNS query/response, TCP, UDP, and IP information.

The agent may also ~~determines~~determine a process associated with the network connection (~~206~~203). One example approach is for the agent to use a kernel network diagnostic API (e.g., netlink_diag) to obtain inode/process information from the kernel. Another example approach is for the agent to scan using netstat (e.g., on ~~-~~/proc/net/tcp, /proc/net/tcp6, /proc/net/udp, and /proc/net/udp6) to obtain sockets and relate them to processes. Information such as socket state (e.g., whether a socket is connected, listening, etc.) can also be collected by the agent.

One way an agent can obtain a mapping between a given inode and a process identifier is to scan within the ~~-~~/proc/pid directory. For each of the processes currently running, the agent examines

each of their file descriptors. If a file descriptor is a match for the inode, the agent can determine that the process associated with the file descriptor owns the inode. Once a mapping is determined between an inode and a process identifier, the mapping is cached. As additional packets are received for the connection, the cached process information is used (rather than a new search being performed).

In some cases, exhaustively scanning for an inode match across every file descriptor may not be feasible (e.g., due to CPU limitations). In various embodiments, searching through file descriptors is accordingly optimized. User filtering is one example of such an optimization. A given socket is owned by a user. Any processes associated with the socket will be owned by the same user as the socket. When matching an inode (identified as relating to a given socket) against processes, the agent can filter through the processes and only examine the file descriptors of processes sharing the same user owner as the socket. In various embodiments, processes owned by root are always searched against (e.g., even when user filtering is employed).

Another example of an optimization is to prioritize searching the file descriptors of certain processes over others. One such prioritization is to search through the subdirectories of /proc/starting with the youngest process. One approximation of such a sort order is to search through -/proc/in reverse order (e.g., examining highest numbered processes first). Higher numbered processes are more likely to be newer (i.e., not long-standing processes), and thus more likely to be associated with new connections (i.e., ones for which inode-process mappings are not already cached). In some cases, the most recently created process may not have the highest process identifier (e.g., due to the kernel wrapping through process identifiers).

Another example prioritization is to query the kernel for an identification of the most recently created process and to search in a backward order through the directories in -/proc/ -(/(e.g., starting at the most recently created process ~~400~~ and working backwards, then wrapping to the highest value (e.g., 32768) and continuing to work backward from there). An alternate approach is for the agent to keep track of the newest process that it has reported information on (e.g., to data aggregator **114**), and begin its search of -/proc/in a forward order starting from the PID of that process.

Another example prioritization is to maintain, for each user actively using - node **116**, a list of the five (or any other number) most recently active processes. Those processes are more likely than other processes (less active, or passive) on - node **116** to be involved with new connections, and can thus be searched first. For many processes, lower valued file descriptors tend to correspond to non-sockets (e.g., stdin, stdout, stderr). Yet another optimization is to preferentially search higher valued file descriptors (e.g., across processes) over lower valued file descriptors (that are less likely to yield matches).

In some cases, while attempting to locate a process identifier for a given inode, an agent may encounter a socket that does not correspond to the inode being matched against and is not already cached. The identity of that socket (and its corresponding inode) can be cached, once discovered, thus removing a future need to search for that pair.

In some cases, a connection may terminate before the agent is able to determine its associated process (e.g., due to a very short-lived connection, due to a backlog in agent processing, etc.). One approach to addressing such a situation is to asynchronously collect information about the connection using the audit kernel API, which streams information to user space. The information collected from the audit API (which can include PID/inode information) can be matched by the agent against pcap/inode information. In some embodiments, the audit API is always used, for all

connections. However, due to CPU utilization considerations, use of the audit API can also be reserved for short/otherwise problematic connections (and/or omitted, as applicable).
Once the agent has determined which process is associated with the network connection (~~206~~203), the agent can then collect additional information associated with the process (~~208~~204). As will be described in more detail below, some of the collected information ~~comprises~~may include attributes of the process (e.g., a process parent hierarchy, and an identification of a binary associated with the process). As will also be described in more detail below, other of the collected information is derived (e.g., session summarization data and hash values).
The collected information is then transmitted (~~210~~205), e.g., by an agent (e.g., agent **112**) to a data aggregator (e.g., data aggregator **114**), which in turn provides ~~it~~ the information to data platform ~~102~~12. In some embodiments, all information collected by an agent ~~is~~may be transmitted (e.g., to a data aggregator and/or to data platform ~~102~~12). In other embodiments, the amount of data transmitted ~~is~~may be minimized (e.g., for efficiency reasons), using various techniques.
One approach to minimizing the amount of data flowing from agents (such as agents installed on nodes **108**) to data platform ~~102~~12 is to use a technique of implicit references with unique keys. The keys can be explicitly used by data platform ~~102~~12 to extract/derive relationships, as necessary, in a data set at a later time, without impacting performance.
~~Depicted in FIG. 3A is initial information collected by an agent about a process. Region **302** indicates the time at which the agent generated event **300**. Region **304** indicates that event **300** pertains to a process. The type identifies a physical collection of related attributes. Other examples of event types that the agent can collect and report (and examples of the types of information included in such reports) are provided below (e.g., in Additional Agent Data Examples).~~
~~Region **306** indicates the process ID (PID) of the process, and region **310** indicates the time the process was started. In region **308** is depicted a hash, generated by the agent to uniquely identify the particular process being reported on in event **300** (e.g., among all other processes whose data is reported to platform **102**). In various embodiments, such hashes (and other hashes, such as are shown at **312**-**316**) are generated pseudo-randomly. The hashes can be reconstructed should the agent crash and restart. An example salt for the hashes is a combination of a local process identifier, start time of the process, and one or more machine attributes.~~
~~Region **318** indicates whether the process is associated with a terminal, an indicator of whether a process is interactive. Region **320** indicates a SHA-256 hash of the entire command line used to start the process, including any variable arguments, and region **322** indicates the path to the executable. Region **324** indicates the effective username of the owner of the process (which can either be the same or different from the username shown in region **328**). As an example, suppose a user logs into node **116** using a first username, and then becomes a different user (e.g., using the setuid system call to inherit the permissions of a second user). Both names can be preserved in regions **324** and **328**, respectively. Finally, region **326** indicates whether the process is running as a container isolation.~~
As previously mentioned, some data collected about a process is constant and does not change over the lifetime of the process (e.g., attributes), and some data changes (e.g., statistical information and other variable information). ~~The data depicted in FIG. 3A is an example of constant data. It~~Constant data can be transmitted (**210**) once, when the agent first becomes aware of the process. And, if any changes to the constant data ~~depicted in FIG. 3A~~ are detected (e.g., a

process changes its parent), a refreshed version of the data ~~depicted in FIG. 3A~~ can be transmitted (**210**) as applicable.

~~FIG. 3B illustrates~~In some examples, an ~~example of~~ agent may collect variable data (e.g., data that ~~can be collected using~~may change over the lifetime of the process ~~shown in FIG. 2. Variable data such as is shown in FIG. 3B~~). In some examples, variable data can be transmitted (**210**) at periodic (or other) intervals. ~~Region **352** indicates a timestamp of when dataset **350** was collected by the agent. The hash depicted in region **354** is~~ Alternatively, variable data may be transmitted in substantially real time as it is collected. In some examples, the ~~same hash that was included in region **308**, and is used within platform **102** to join process creation time attributes (e.g., shown in FIG. 3A) with runtime attributes (e.g., shown in FIG. 3B) to construct a full dataset. Region **356** indicates~~variable data may indicate a thread count for ~~the~~a process~~. Region **358** indicates the~~, a total virtual memory used by the process~~. Region **360** indicates~~, the total resident memory used by the process~~. Region **362** indicates~~, the total time spent by the process executing in user space~~. Region **364** indicates~~, and/or the total time spent by the process executing in kernel space. In some examples, the data may include a hash that may be used within data platform **12** to join process creation time attributes with runtime attributes to construct a full dataset.

~~C. Additional Agent Data Examples~~

Below are additional examples of data that an agent, such as agent **112**, can collect and provide to data platform ~~**102**~~**12**.

1. User Data

Core User Data: user name, UID (user ID), primary group, other groups, home directory.

Failed Login Data: IP address, hostname, username, count.

User Login Data: user name, hostname, IP address, start time, TTY (terminal), UID (user ID), GID (group ID), process, end time.

2. Machine Data

Dropped Packet Data: source IP address, destination IP address, destination port, protocol, count.

Machine Data: hostname, domain name, architecture, kernel, kernel release, kernel version, OS, OS version, OS description, CPU, memory, model number, number of cores, last boot time, last boot reason, tags (e.g., Cloud provider tags such as AWS, GCP, or Azure tags), default router, interface name, interface hardware address, interface IP address and mask, promiscuous mode.

3. Network Data

Network Connection Data: source IP address, destination IP address, source port, destination port, protocol, start time, end time, incoming and outgoing bytes, source process, destination process, direction of connection, histograms of packet length, inter packet delay, session lengths, etc.

Listening Ports in Server: source IP address, port number, protocol, process.

Dropped Packet Data: source IP address, destination IP address, destination port, protocol, count.

Arp Data: source hardware address, source IP address, destination hardware address, destination IP address.

DNS Data: source IP address, response code, response string, question (request), packet length, final answer (response).

4. Application Data

Package Data: exe path, package name, architecture, version, package path, checksums (MD5, SHA-1, SHA-256), size, owner, owner ID.

Application Data: command line, PID (process ID), start time, UID (user ID), EUID (effective UID), PPID (parent process ID), PGID (process group ID), SID (session ID), exe path, username, container ID.
5. Container Data
Container Image Data: image creation time, parent ID, author, container type, repo, (AWS) tags, size, virtual size, image version.
Container Data: container start time, container type, container name, container ID, network mode, privileged, PID mode, IP addresses, listening ports, volume map, process ID.
6. File Data
File path, file data hash, symbolic links, file creation data, file change data, file metadata, file mode.

~~D. Additional Information about Containers~~

As mentioned above, an agent, such as agent **112**, can be deployed in a container (e.g., a Docker container), and can also be used to collect information about containers. Collection about a container can be performed by an agent irrespective of whether the agent is itself deployed in a container or not (as the agent can be deployed in a container running in a privileged mode that allows for monitoring).
Agents can discover containers (e.g., for monitoring) by listening for container create events (e.g., provided by Docker), and can also perform periodic ordered discovery scans to determine whether containers are running on a node. When a container is discovered, the agent can obtain attributes of the container, e.g., using standard Docker API calls (e.g., to obtain IP addresses associated with the container, whether there's a server running inside, what port it is listening on, associated PIDs, etc.). Information such as the parent process that started the container can also be collected, as can information about the image (which comes from the Docker repository).
In various embodiments, agents may use namespaces to determine whether a process is associated with a container ~~or not~~. Namespaces are a feature of the Linux kernel that can be used to isolate resources of a collection of processes. Examples of namespaces include process ID (PID) namespaces, network namespaces, and user namespaces. Given a process, the agent can perform a fast lookup to determine whether the process is part of the namespace the container claims to be its namespace.

~~E. Data Reporting~~

As mentioned ~~above~~, agents can be configured to report certain types of information (e.g., attribute information) once, when the agent first becomes aware of a process. In various embodiments, such static information is not reported again (or is reported once a day, every twelve hours, etc.), unless it changes (e.g., a process changes its parent, changes its owner, or a SHA-1 of the binary associated with the process changes).
In contrast to static/attribute information, certain types of data change constantly (e.g., network-related data). In various embodiments, agents are configured to report a list of current connections every minute (or other appropriate time interval). In that connection list will be connections that started in that minute interval, connections that ended in that minute interval, and connections that were ongoing throughout the minute interval (e.g., a one minute slice of a one hour connection).
In various embodiments, agents are configured to collect/compute statistical information about connections (e.g., at the one minute level of granularity~~).~~ and or at any other time interval).
Examples of such information include, for the time interval, the number of bytes transferred, and in which direction. Another example of information collected by an agent about a connection is

the length of time between packets. For connections that span multiple time intervals (e.g., a seven minute connection), statistics ~~are~~may be calculated for each minute of the connection. Such statistical information (for all connections) can be reported (e.g., to a data aggregator) once a minute.

In various embodiments, agents are also configured to maintain histogram data for a given network connection, and provide ~~it (~~the histogram data (e.g., in the Apache Avro data exchange format) under the Connection event type data. Examples of such histograms include: 1. a packet length histogram (packet_len_hist), which characterizes network packet distribution; 2. a session length histogram (session_len_hist), which characterizes a network session length; 3. a session time histogram (session_time_hist), which characterizes a network session time; and 4. a session switch time histogram (session_switch_time_hist), which characterizes network session switch time (i.e., incoming→outgoing and vice versa). ~~FIGS. 3C and 3D depict, side-by-side, examples of full outgoing (3C) and incoming (3D) network~~ For example, histogram data ~~sets, including each~~may include one or more of the ~~four histograms. Each of the histograms shown in FIGS. 3C and 3D includes the~~ following fields: 1. count, which provides a count of the elements in the sampling; 2. sum, which provides a sum of elements in the sampling; 3. max, which provides the highest value element in the sampling; 4. std_dev, which provides the standard deviation of elements in the sampling; and 5. buckets, which provides a discrete sample bucket distribution of sampling data (if applicable).

For some protocols (e.g., HTTP), typically, a connection is opened, a string is sent, a string is received, and the connection is closed. For other protocols (e.g., NFS), both sides of the connection engage in a constant chatter. Histograms allow data platform ~~102~~12 to model application behavior (e.g., using machine learning techniques), for establishing baselines, and for detecting deviations. As one example, suppose that a given HTTP server typically sends/receives 1,000 bytes (in each direction) whenever a connection is made with it. If a connection generates 500 bytes of traffic, or 2,000 bytes of traffic, such connections would be considered within the typical usage pattern of the server. Suppose, however, that a connection is made that results in **10**G of traffic. Such a connection is anomalous and can be flagged accordingly.

~~F. Storing Data Collected by Agents~~

Returning to FIG. ~~1~~ 1D, as previously mentioned, data aggregator **114** ~~is~~may be configured to provide information (e.g., collected from nodes **108** by agents) to data platform ~~102. And, data~~ **12**. Data aggregator **128** ~~is~~may be similarly configured to provide information to data platform ~~102~~12. As shown in FIG. ~~1~~ 1D, both aggregator **114** and aggregator **128** may connect to a load balancer **130**, which accepts connections from aggregators (and/or as applicable, agents), as well as other devices, such as computer **126** (e.g., when it communicates with web app **120**), and supports fair balancing. In various embodiments, load balancer **130** is a reverse proxy that load balances accepted connections internally to various microservices (described in more detail below), allowing for services provided by data platform ~~102~~12 to scale up as more agents are added to the environment and/or as more entities subscribe to services provided by data platform ~~102~~12. Example ways to implement load balancer **130** include, but are not limited to, using HaProxy, using nginx, and using elastic load balancing (ELB) services made available by Amazon.

Agent service **132** is a microservice that is responsible for accepting data collected from agents (e.g., provided by aggregator **114**). In various embodiments, agent service **132** uses a standard secure protocol, such as HTTPS to communicate with aggregators (and as applicable agents), and receives data in an appropriate format such as Apache Avro. When agent

service **132** receives an incoming connection, it can perform a variety of checks, such as to see whether the data is being provided by a current customer, and whether the data is being provided in an appropriate format. If the data is not appropriately formatted (and/or is not provided by a current customer), it ~~is~~may be rejected.

If the data is appropriately formatted, agent service **132** ~~facilitates~~may facilitate copying the received data to a streaming data stable storage using a streaming service (e.g., ~~using~~ Amazon Kinesis ~~(134)).~~and/or any other suitable streaming service. Once the ingesting into ~~Kinesis~~the streaming service is complete, service **132** ~~sends~~may send an acknowledgement to the data provider (e.g., data aggregator **114**). If the agent does not receive such an acknowledgement, it is configured to retry sending the data to data platform ~~102~~**12**. One way to implement agent service **132** is as a REST API server framework (e.g., Java DropWizard), configured to communicate with Kinesis (e.g., using a Kinesis library).

In various embodiments, data platform ~~102~~**12** uses one or more streams (e.g., Kinesis streams ~~(134~~) for all incoming customer data (e.g., including data provided by- data aggregator **114** and data aggregator **128**), and the data is sharded based on the node (also referred to herein as a "machine") that originated the data (e.g.,- node **116** vs. node **122**), with each node having a globally unique identifier within data platform ~~102~~**12**. Multiple instances of agent service **132** can write to multiple shards.

Kinesis is a streaming service with a limited period (e.g., 1-7 days). To persist data longer than a day, ~~it is~~the data may be copied to long term storage- **42** (e.g., S3). ~~S3 Loader~~Data loader **136** is a microservice that is responsible for picking up data from- a data stream (e.g., a Kinesis stream ~~134~~) and persisting it in- ~~S3 (138).~~ long term storage **42**. In one example embodiment, files collected by ~~S3 Loader~~data loader **136** from the Kinesis stream are placed into one or more buckets, and segmented using a combination of a customer identifier and time slice. Given a particular time segment, and a given customer identifier, the corresponding file (stored in ~~S3~~long term storage) contains five minutes (or another appropriate time slice) of data collected at that specific customer from all of the customer's nodes. ~~S3 Loader~~Data loader **136** can be implemented in any appropriate programming language, such as Java or C, and can be configured to use a Kinesis library to interface with Kinesis. In various embodiments, ~~S3 Loader~~data loader **136** uses the Amazon Simple Queue Service (SQS) (e.g., to alert DB ~~Loader~~loader **140** that there is work for it to do).

DB ~~Loader~~loader **140** is a microservice that is responsible for loading data into an- appropriate ~~database service (142),~~data store **30**, such as SnowflakeDB or Amazon Redshift, using individual per-customer databases. In particular, DB ~~Loader~~loader **140** is configured to periodically load data into a set of raw tables from files created by ~~S3 Loader~~data loader **136** as per above. DB ~~Loader~~loader **140** manages throughput, errors, etc., to make sure that data is loaded consistently and continuously. Further, DB ~~Loader~~loader **140** can read incoming data and load into ~~database 142~~data store **30** data that is not already present in- ~~database 142's~~ tables- of data store **30** (also referred to herein as a database). DB ~~Loader~~loader **140** can be implemented in any appropriate programming language, such as Java or C, and an SQL framework such as jOOQ (e.g., to manage SQLs for insertion of data), and SQL/JDBC libraries. In some examples, DB ~~Loader~~loader **140** ~~also uses~~may use Amazon S3 and Amazon Simple Queue Service (SQS) to manage files being transferred to and from- ~~the database (e.g., Snowflake).~~ data store **30**.

Customer data included in ~~database 142~~data store **30** can be augmented with data from additional data sources, such as AWS- ~~Cloud Trail Analyzer~~ CloudTrail and/or other types of external tracking services. To this end, data platform may include a tracking service analyzer **144**, which

is another microservice. ~~AWS Cloud Trail Analyzer~~ Tracking service analyzer **144** ~~pulls data using~~may pull data from an external tracking service (e.g., Amazon ~~Cloudtrail~~CloudTrail) for each applicable customer account, as soon as the data is available. ~~AWS Cloud Trail Analyzer~~ Tracking service analyzer **144** ~~normalizes~~may normalize the ~~Cloudtrail~~tracking data as applicable, so that it can be inserted into ~~database **142**~~data store **30** for later querying/analysis. ~~AWS Cloud Trail Analyzer~~ Tracking service analyzer **144** can be written in any appropriate programming language, such as Java or C. ~~AWS Cloud Trail Analyzer~~ Tracking service analyzer **144** also makes use of SQL/JDBC libraries to interact with ~~database **142**~~data store **30** to insert/query data.

~~II. Polygraphs~~

~~A. Polygraph Overview~~

As ~~previously mentioned,~~ described herein, data platform ~~**102**~~**12** can model activities that occur within datacenters, such as datacenters **104** and **106**. The model ~~is~~may be stable over time, and differences, even subtle ones (e.g., between a current state of the datacenter and the model) can be surfaced. The ability to surface such anomalies can be particularly ~~important~~beneficial in datacenter environments where rogue employees and/or external attackers may operate slowly (e.g., over a period of months), hoping that the elastic nature of typical resource use (e.g., virtualized servers) will help conceal their nefarious activities.

Using techniques described herein, data platform ~~**102**~~**12** can automatically discover entities (which may implement compute assets **16**) deployed in a given datacenter. Examples of entities include workloads, applications, processes, machines, virtual machines, containers, files, IP addresses, domain names, and users. The entities ~~are~~may be grouped together logically (into analysis groups) based on behaviors, and temporal behavior baselines can be established. In particular, using techniques described herein, periodic graphs can be constructed (also referred to herein as polygraphs), in which the nodes are applicable logical entities, and the edges represent behavioral relationships between the logical entities in the graph. Baselines can be created for every node and edge.

Communication (e.g., between applications/nodes) is one example of a behavior. A model of communications between processes is an example of a behavioral model. As another example, the launching of applications is another example of a behavior that can be modeled. The baselines ~~are~~may be periodically updated (e.g., hourly) for every entity. Additionally or alternatively, the baselines may be continuously updated in substantially real-time as data is collected by agents. Deviations from the expected normal behavior can then be detected and automatically reported (e.g., as anomalies or threats detected). Such deviations may be due to a desired change, a misconfiguration, or malicious activity. As applicable, data platform ~~**102**~~**12** can score the detected deviations (e.g., based on severity and threat posed). Additional examples of analysis groups include models of machine communications, models of privilege changes, and models of insider behaviors (monitoring the interactive behavior of human users as they operate within the datacenter).

Two example types of information collected by agents ~~(described in more detail above)~~ are network level information and process level information. As previously mentioned, agents may collect information about every connection involving their respective nodes. And, for each connection, information about both the server and the client ~~is~~may be collected (e.g., using the connection-to-process identification techniques described above). DNS queries and responses ~~are~~may also be collected. The DNS query information can be used in logical entity graphing (e.g., collapsing many different IP addresses to a single service—e.g., s3.amazon.com).

Examples of process level information collected by agents include attributes (user ID, effective user ID, and command line). Information such as what user/application is responsible for launching a given process and the binary being executed (and its SHA-256 values) ~~is~~may also be provided by agents.

The dataset collected by agents across a datacenter can be very large, and many resources (e.g., virtual machines, IP addresses, etc.) are recycled very quickly. For example, an IP address and port number used at a first point in time by a first process on a first virtual machine may very rapidly be used (e.g., an hour later) by a different process/virtual machine.

A dataset (and elements within it) can be considered at both a physical level, and a logical level, as illustrated in FIG. ~~4~~ 2B. In particular, FIG. ~~4~~ 2B illustrates an example 5-tuple of data ~~(402)~~ **210** collected by an agent, represented physically (~~414~~**216**) and logically (~~416~~**217**). The 5-tuple includes a source address ~~(404),~~ **211,** a source port ~~(406),~~ **212,** a destination address ~~(408),~~ **213,** a destination port ~~(410),~~ **214,** and a protocol ~~(412),~~ **215.** In some cases, port numbers (e.g., ~~406, 410~~**212, 214**) may be indicative of the nature of a connection (e.g., with certain port usage standardized). However, in many cases, and in particular in datacenters, port usage is ephemeral. For example, a Docker container can listen on an ephemeral port, which is unrelated to the service it will run. When another Docker container starts (for the same service), the port may well be different. Similarly, particularly in a virtualized environment, IP addresses may be recycled frequently (and are thus also potentially ephemeral) or could be NATed, which makes identification difficult.

A physical representation of the 5-tuple is depicted in region ~~414~~**216**. A process ~~418~~**218** (executing on machine ~~420~~**219**) has opened a connection to machine ~~422~~**220**. In particular, process ~~418~~**218** is in communication with process ~~424~~**221**. Information such as the number of packets exchanged between the two machines over the respective ports can be recorded.

As previously mentioned, in a datacenter environment, portions of the 5-tuple may change—potentially frequently—but still be associated with the same behavior. Namely, one application (e.g., Apache) may frequently be in communication with another application (e.g., Oracle), using ephemeral datacenter resources. Further, either/both of Apache and Oracle may be multi-homed. This can lead to potentially thousands of 5-tuples (or more) that all correspond to Apache communicating with Oracle within a datacenter. For example, Apache could be executed on a single machine, and could also be executed across fifty machines, which are variously spun up and down (with different IP addresses each time). An alternate representation of the 5-tuple ~~shown in region 402~~of data **210** is depicted in region ~~416~~**217**, and is logical. The logical representation of the 5-tuple aggregates the 5-tuple (along with other connections between Apache and Oracle having other 5-tuples) as logically representing the same connection. By aggregating data from raw physical connection information into logical connection information, using techniques described herein, a size reduction of six orders of magnitude in the data set can be achieved.

FIG. ~~5~~ 2C depicts a portion of a logical polygraph. Suppose a datacenter has seven instances of the application update_engine ~~(502),~~ **225,** executing as seven different processes on seven different machines, having seven different IP addresses, and using seven different ports. The instances of update_engine variously communicate with update.core-os.net ~~(504),~~ **226,** which may have a single IP address or many IP addresses itself, over the one hour time period represented in the polygraph. In the example shown in FIG. ~~5~~ 2C, update_engine is a client, connecting to the server update.core-os.net, as indicated by arrow ~~508~~**228**.

Behaviors of the seven processes are clustered together, into a single summary. As indicated in region ~~506~~227, statistical information about the connections is also maintained (e.g., number of connections, histogram information, etc.). A polygraph such as is depicted in FIG. ~~5~~ 2C can be used to establish a baseline of behavior (e.g., at the one-hour level), allowing for the future detection of deviations from that baseline. As one example, suppose that statistically an update_engine instance transmits data at 11 bytes per second. If an instance were instead to transmit data at 1000 bytes per second, such behavior would represent a deviation from the baseline and could be flagged accordingly. Similarly, changes that are within the baseline (e.g., an eighth instance of update_engine appears, but otherwise behaves as the other instances; or one of the seven instances disappears) are not flagged as anomalous. Further, datacenter events, such as failover, autobalancing, and A-B refresh are unlikely to trigger false alarms in a polygraph, as at the logical level, the behaviors remain the same.

In various embodiments, polygraph data is maintained for every application in a datacenter, and such polygraph data can be combined to make a single datacenter view across all such applications. FIG. ~~6~~ 2D illustrates a portion of a polygraph for a service that evidences more complex behaviors than are depicted in FIG. ~~5~~ 2C. In particular, FIG. ~~6~~ 2D illustrates the behaviors of S3 as a service (as used by a particular customer datacenter). Clients within the datacenter variously connect to the S3 service using one of five fully qualified domains (listed in region ~~602~~230). Contact with any of the domains is aggregated as contact with S3 (as indicated in region ~~604~~231). Depicted in region ~~606~~232 are various containers which (as clients) connect with S3. Other containers (which do not connect with S3) are not included. As with the polygraph portion depicted in FIG. ~~5~~ 2C, statistical information about the connections is known and summarized, such as the number of bytes transferred, histogram information, etc.

FIG. ~~7~~ 2E illustrates a communication polygraph for a datacenter. In particular, the polygraph indicates a one hour summary of approximately 500 virtual machines, which collectively run one million processes, and make 100 million connections in that hour. As illustrated in FIG. ~~7~~ 2E, a polygraph represents a drastic reduction in size (e.g., from tracking information on 100 million connections in an hour, to a few hundred nodes and a few hundred edges). Further, as a datacenter scales up (e.g., from using 10 virtual machines to 100 virtual machines as the datacenter uses more workers to support existing applications), the polygraph for the datacenter will tend to stay the same size (with the 100 virtual machines clustering into the same nodes that the 10 virtual machines previously clustered into). As new applications are added into the datacenter, the polygraph ~~will~~may automatically scale to include behaviors involving those applications.

In the particular polygraph shown in FIG. ~~7~~ 2E, nodes generally correspond to workers, and edges correspond to communications the workers engage in (with connection activity being the behavior modeled in polygraph ~~700~~235). Another example polygraph could model other behavior, such as application launching. The communications graphed in FIG. ~~7~~ 2E include traffic entering the datacenter, traffic exiting the datacenter, and traffic that stays wholly within the datacenter (e.g., traffic between workers). One example of a node included in polygraph ~~700~~235 is the sshd application, depicted as node ~~702~~236. As indicated in FIG. ~~7~~ 2E, 421 instances of sshd were executing during the one hour time period of data represented in polygraph ~~700~~235. As indicated in region ~~704~~237, nodes within the datacenter communicated with a total of 1349 IP addresses outside of the datacenter (and not otherwise accounted for, e.g., as belonging to a service such as Amazon AWS ~~(706)~~ 238 or Slack ~~(708))~~ 239).

~~B. Interacting with Polygraphs~~

In the following examples, suppose that ~~Bob~~user B, an administrator of datacenter **106**, is interacting with data platform ~~10~~**212** to view visualizations of polygraphs in a web browser (e.g., as served to ~~him~~user B via web app **120**). One type of polygraph ~~Bob~~user B can view is an application-communication polygraph, which indicates, for a given one hour window~~,~~ (or any other suitable time interval), which applications communicated with which other applications. Another type of polygraph ~~Bob~~user B can view is an application launch polygraph. ~~Bob~~User B can also view graphs related to user behavior, such as an insider behavior graph which tracks user connections (e.g., to internal and external applications, including chains of such behavior), a privilege change graph which tracks how privileges change between processes, and a user login graph, which tracks which (logical) machines a user logs into.

FIG. ~~8~~ 2F illustrates an example of an application-communication polygraph for a datacenter (e.g., datacenter **106**) for the one hour period of 9 am-10 am on June 5. The time slice currently being viewed is indicated in region **802240**. If ~~Bob~~user B clicks his mouse in region **804,** ~~Bob~~**241**, user B will be shown a representation of the application-communication polygraph as generated for the following hour (10 am-11 am on June 5).

FIG. ~~9~~ 2G depicts what is shown in ~~Bob's~~user B's browser after he has clicked on region **804241**, and has further clicked on region **806242**. The selection in region **806242** turns on and off the ability to compare two time intervals to one another. ~~Bob~~User B can select from a variety of options when comparing the 9 am-10 am and 10 am-11 am time intervals. By clicking region **904, Bob248**, user B will be shown the union of both graphs (i.e., any connections that were present in either time interval). By clicking region **906, Bob249**, user B will be shown the intersection of both graphs (i.e., only those connections that were present in both time intervals).

As shown in FIG. ~~9, Bob~~ 2G, user B has elected to click on region **908250**, which depicts connections that are only present in the 9 am-10 am polygraph in ~~purple (910),~~a first color **251**, and depicts connections that are only present in the 10 am-11 am polygraph in ~~blue (912).~~a second color **252**. Connections present in both polygraphs are omitted from display. As one example, in the 9 am-10 am polygraph (corresponding to connections made during the 9 am-10 am time period at datacenter **106**), a connection was made by a server to sshd (**914253**) and also to systemd (**916254**). Both of those connections ended prior to 10 am and are thus depicted in ~~purple~~the first color. As another example, in the 10 am-11 am polygraph (corresponding to connections made during the 10 am-11 am time period at datacenter **106**), a connection was made from a known bad external IP to nginx (**918255**). The connection was not present during the 9 am-10 am time slice and thus is depicted in ~~blue~~the second color. As yet another example, two different connections were made to a Slack service between 9 am and 11 am. However, the first was made by a first client during the 9 am-10 am time slice (**920256**) and the second was made by a different client during the 10 am-11 am slice (**922257**), and so the two connections are depicted respectively in ~~purple~~the first and second colors and blue.

Returning to the polygraph depicted in FIG. ~~8~~ 2F, suppose ~~Bob~~user B enters “etcd” into the search box located in region **810. Bob244**. User B will then be presented with the interface illustrated in FIG. ~~10~~ 2H. As shown in FIG. ~~10~~ 2H, three applications containing the term “etcd” were engaged in communications during the 9 am-10 am window. One application is etcdctl, a command line client for etcd. As shown in FIG. ~~10~~ 2H, a total of three different etcdctl processes were executed during the 9 am-10 am window, and were clustered together (**1002260**). FIG. ~~10~~ 2H also depicts two different clusters that are both named etcd2. The first cluster includes (for the 9 am-10 am window) five members (**1004261**) and the second cluster includes (for the same

window) eight members (~~1006~~262). The reason for these two distinct clusters is that the two groups of applications behave differently (e.g., they exhibit two distinct sets of communication patterns). Specifically, the instances of etcd2 in cluster ~~1004~~261 only communicate with locksmithctl (~~1008~~263) and other etcd2 instances (in both clusters ~~1004~~261 and ~~1006~~262). The instances of etcd2 in cluster ~~1006~~262 communicate with additional entities, such as etcdctl and Docker containers. As desired, ~~Bob~~user B can click on one of the clusters (e.g., cluster ~~1004~~261) and be presented with summary information about the applications included in the cluster, as is shown in FIG. ~~11~~ 2I (e.g., in region ~~1102). He~~265). User B can also double click on a given cluster (e.g., cluster ~~1004~~261) to see details on each of the individual members of the cluster broken out.

Suppose ~~Bob~~user B now clicks on region ~~812~~245 of the interface shown in FIG. ~~8. Bob~~ 2F. User B will then be shown an application launch polygraph. Launching an application is another example of a behavior. The launch polygraph models how applications are launched by other applications. FIG. ~~12~~ 2J illustrates an example of a portion of a launch polygraph. In particular, ~~Bob~~user B has typed "find" into region ~~1202~~266, to see how the "find" application is being launched. As shown in FIG. ~~12~~ 2J, in the launch polygraph for the 10 am-11 am time period, find applications (~~1204~~267) are always launched by bash (~~1206~~268), which is in turn always launched by systemd (~~1208~~269). If find is launched by a different application, this would be anomalous behavior.

FIG. ~~13~~ 2K illustrates another example of a portion of an application launch polygraph. In FIG. ~~13, Bob~~ 2K, user B has searched (~~1302~~270) for "python ma" to see how "python marathon_lb" (~~1304~~271) is launched. As shown in FIG. ~~13~~ 2K, in each case (during the one hour time slice of 10 am-11 am), python marathon_lb is launched as a result of a chain of the same seven applications each time. If python marathon_lb is ever launched in a different manner, this indicates anomalous behavior. The behavior could be indicative of malicious activities, but could also be due to other reasons, such as a misconfiguration, a performance-related issue, and/or a failure, etc.

Suppose ~~Bob~~user B now clicks on region ~~814~~246 of the interface shown in FIG. ~~8. Bob~~ 2F. User B will then be shown an insider behavior graph. The insider behavior graph tracks information about behaviors such as processes started by a user interactively using protocols such as ssh or telnet, and any processes started by those processes. As one example, suppose an administrator logs into a first virtual machine in datacenter **106** (e.g., using sshd via an external connection he makes from a hotel), using a first set of credentials (e.g., ~~charlie.smith~~first.last@example.com and an appropriate password). From the first virtual machine, the administrator connects to a second virtual machine (e.g., using the same credentials), then uses the sudo command to change identities to those of another user, and then launches a program. ~~Graphs~~graphs built by data platform ~~102~~212 can be used to associate the administrator with each of his actions, including launching the program using the identity of another user.

FIG. ~~14~~ 2L illustrates an example of a portion of an insider behavior graph. In particular, in FIG. ~~14, Bob~~ 2L, user B is viewing a graph that corresponds to the time slice of 3 pm-4 pm on June 1. FIG. ~~14~~ 2L illustrates the internal/external applications that users connected to during the one hour time slice. If a user typically communicates with particular applications, that information will become part of a baseline. If the user deviates from his baseline behavior (e.g., using new applications, or changing privilege in anomalous ways), such anomalies can be surfaced.

FIG. ~~15~~ 2M illustrates an example of a portion of a privilege change graph, which identifies how privileges are changed between processes. Typically, when a user launches a process (e.g.,

"~~ls~~Is"), the process inherits the same privileges that the user has. And, while a process can have fewer privileges than the user (i.e., go down in privilege), it is rare (and generally undesirable) for a user to escalate in privilege. Information included in the privilege change graph can be determined by examining the parent of each running process, and determining whether there is a match in privilege between the parent and the child. If the privileges are different, a privilege change has occurred (whether a change up or a change down). The application ntpd is one rare example of a scenario in which a process escalates (~~1502~~272) to root, and then returns back (~~1504~~273). The sudo command is another example (e.g., used by an administrator to temporarily have a higher privilege). As with the other examples, ntpd's privilege change actions, and the legitimate actions of various administrators (e.g., using sudo) will be incorporated into a baseline model by data platform ~~102~~12. When deviations occur, such as where a new application that is not ntpd escalates privilege, or where an individual that has not previously/does not routinely use sudo does so, such behaviors can be identified as anomalous.

FIG. ~~16~~ 2N illustrates an example of a portion of a user login graph, which identifies which users log into which logical nodes. Physical nodes (whether bare metal or virtualized) are clustered into a logical machine cluster, for example, using yet another graph, a machine-server graph, an example of which is shown in FIG. ~~17~~ 2O. For each machine, a determination is made as to what type of machine it is, based on what kind(s) of workflows it runs. As one example, some machines run as master nodes (having a typical set of workflows they run, as master nodes) and can thus be clustered as master nodes. Worker nodes are different from master nodes, for example, because they run Docker containers, and frequently change as containers move around. Worker nodes can similarly be clustered.

~~Additional information regarding use of graphs such as are depicted in FIGS. 14-17 is provided in Section IV below.~~

~~C. Polygraph Construction~~

As previously mentioned, the polygraph depicted in FIG. ~~7~~ 2E corresponds to activities in a datacenter in which, in a given hour, approximately 500 virtual machines collectively run one million processes, and make 100 million connections in that hour. The polygraph represents a drastic reduction in size (e.g., from tracking information on 100 million connections in an hour, to a few hundred nodes and a few hundred edges). Using techniques described herein, such a polygraph can be constructed (e.g., using commercially available computing infrastructure) in less than an hour (e.g., within a few minutes). Thus, ongoing hourly snapshots of a datacenter can be created within a two hour moving window (i.e., collecting data for the time period 8 am-9 am, while also generating a snapshot for the time previous time period 7 am-8 am). The following ~~section~~ describes various example infrastructure that can be used in polygraph construction, and also describes various techniques that can be used to construct polygraphs.

~~1. Example Infrastructure~~

Returning to FIG. ~~1~~ 1D, embodiments of data platform ~~102 are~~12 may be built using ~~AWS~~any suitable infrastructure as a service (IaaS) (e.g., AWS). For example, data platform ~~102~~12 can use Simple Storage Service (S3) for data storage, Key Management Service (KMS) for managing secrets, Simple Queue Service (SQS) for managing messaging between applications, Simple Email Service (SES) for sending emails, and Route- **53**- for managing DNS. Other infrastructure tools can also be used. Examples include: orchestration tools (e.g., Kubernetes or Mesos/Marathon), service discovery tools (e.g., Mesos-DNS), service load balancing tools (e.g., marathon-LB), container tools (e.g., Docker or rkt), log/metric tools (e.g., collectd, fluentd,

kibana, etc.), big data processing systems (e.g., Spark, Hadoop, AWS Redshift, Snowflake etc.), and distributed key value stores (e.g., Apache Zookeeper or etcd2).

As previously mentioned, in various embodiments, data platform ~~102 makes~~12 may make use of a collection of microservices. Each microservice can have multiple instances, and ~~is~~may be configured to recover from failure, scale, and distribute work amongst various such instances, as applicable. For example, microservices are auto-balancing for new instances, and can distribute workload if new instances are started or existing instances are terminated. In various embodiments, microservices ~~are~~may be deployed as self-contained Docker containers. A Mesos-Marathon or Spark framework can be used to deploy the microservices (e.g., with Marathon monitoring and restarting failed instances of microservices as needed). The service etcd2 can be used by microservice instances to discover how many peer instances are running, and used for calculating a hash-based scheme for workload distribution. Microservices ~~are~~may be configured to publish various health/status metrics to either an SQS queue, or etcd2, as applicable. ~~And~~In some examples, Amazon DynamoDB can be used for state management.

Additional information on various microservices used in embodiments of data platform ~~102~~12 is provided below.

~~a. GraphGen~~

~~GraphGen (~~Graph generator **146**~~)~~ is a microservice that ~~is~~may be responsible for generating raw behavior graphs on a per customer basis periodically (e.g., once an hour). In particular, ~~GraphGen~~graph generator **146** ~~generates~~may generate graphs of entities (as the nodes in the graph) and activities between entities (as the edges). In various embodiments, ~~GraphGen~~graph generator **146** also performs other functions, such as aggregation, enrichment (e.g., geolocation and threat), reverse DNS resolution, TF-IDF based command line analysis for command type extraction, parent process tracking, etc.

~~GraphGen~~Graph generator **146** ~~performs~~may perform joins on data collected by the agents, so that both sides of a behavior are linked. For example, suppose a first process on a first virtual machine (e.g., having a first IP address) communicates with a second process on a second virtual machine (e.g., having a second IP address). Respective agents on the first and second virtual machines ~~will~~may each report information on their view of the communication (e.g., the PID of their respective processes, the amount of data exchanged and in which direction, etc.). When ~~GraphGen~~graph generator performs a join on the data provided by both agents, the graph will include a node for each of the processes, and an edge indicating communication between them (as well as other information, such as the directionality of the communication—i.e., which process acted as the server and which as the client in the communication).

In some cases, connections are process to process (e.g., from a process on one virtual machine within ~~ACME~~the cloud environment associated with entity A to another process on a virtual machine within ~~ACME),~~the cloud environment associated with entity A). In other cases, a process may be in communication with a node (e.g., outside of ~~ACME~~entity A) which does not have an agent deployed upon it. As one example, a node within ~~ACME~~entity A might be in communication with node **172**, outside of ~~ACME~~entity A. In such a scenario, communications with node **172** are modeled (e.g., by ~~GraphGen~~graph generator **146**) using the IP address of node **172**. Similarly, where a node within ~~ACME~~entity A does not have an agent deployed upon it, the IP address of the node can be used by ~~GraphGen~~graph generator in modeling.

Graphs created by ~~GraphGen~~graph generator **146** ~~are~~may be written to ~~database **142**~~data store **30** and cached for further processing. ~~The~~A graph ~~is~~may be a summary of all activity that happened in ~~the~~a particular time interval. As each graph corresponds to a distinct period of time,

different rows can be aggregated to find summary information over a larger timestamp. ~~And~~In some examples, picking two different graphs from two different timestamps can be used to compare different periods. If necessary, ~~GraphGen~~graph generator can parallelize its workload (e.g., where its backlog cannot otherwise be handled within a particular time period, such as an hour, or if is required to process a graph spanning a long time period).

~~GraphGen~~ Graph generator **146** can be implemented in any appropriate programming language, such as Java or C, and machine learning libraries, such as Spark's MLLib. Example ways that ~~GraphGen~~graph generator computations can be implemented include using SQL or Map-R, using Spark or Hadoop.

~~b. sshTracker~~

~~SshTracker (~~SSH tracker **148**~~)~~ is a microservice that ~~is~~may be responsible for following ssh connections and process parent hierarchies to determine trails of user ssh activity. Identified ssh trails are placed by the ~~sshTracker~~ SSH tracker **148** into ~~database **142**~~data store **30** and cached for further processing.

~~SshTracker~~SSH tracker **148** can be implemented in any appropriate programming language, such as Java or C, and machine libraries, such as Spark's MLLib. Example ways that ~~sshTracker~~SSH tracker computations can be implemented include using SQL or Map-R, using Spark or Hadoop.

~~c. ThreatAggr~~

~~ThreatAggr (~~Threat aggregator **150**~~)~~ is a microservice that ~~is~~may be responsible for obtaining third party threat information from various applicable sources, and making it available to other microservices. Examples of such information include reverse DNS information, GeoIP information, lists of known bad domains/IP addresses, lists of known bad files etc. As applicable, the threat information is normalized before insertion into ~~database **142**. ThreatAggr~~ data store **30**. Threat aggregator **150** can be implemented in any appropriate programming language, such as Java or C, using SQL/JDBC libraries to interact with ~~database **142**~~data store **30** (e.g., for insertions and queries).

~~d. Hawkeye~~

~~Hawkeye (~~Scheduler **152**~~)~~ is a microservice that ~~acts~~may act as a scheduler and ~~can~~that may run arbitrary jobs organized as a directed graph. ~~Hawkeye~~ In some examples, scheduler **152** ensures that all jobs for all customers are able to run during at a given time interval (e.g., every hour~~. It handles~~). Scheduler **152** may handle errors and retrying for failed jobs, ~~tracks~~track dependencies, ~~manages~~manage appropriate resource levels, and ~~scales~~/or scale jobs as needed.

~~Hawkeye~~ Scheduler **152** can be implemented in any appropriate programming language, such as Java or C. A variety of components can also be used, such as open source scheduler frameworks (e.g., Airflow), or AWS services (e.g., the AWS Data pipeline) which can be used for managing schedules.

~~e. GBM~~

Graph Behavior Modeler (GBM) ~~(~~ **154**~~)~~ is a microservice that ~~computes Polygraphs.~~may compute polygraphs. In particular, ~~the~~ GBM **154** can be used to find clusters of nodes in a graph that should be considered similar based on some set of their properties and relationships to other nodes. As ~~will be~~ described ~~in more detail below~~herein, the clusters and their relationships can be used to provide visibility into a datacenter environment without requiring user specified labels. ~~The~~ GBM ~~tracks~~ **154** may track such clusters over time persistently, allowing for changes to be detected and alerts to be generated.

~~The~~ GBM ~~takes~~ **154** may take as input a raw graph (e.g., as generated by ~~GraphGen~~graph generator **146**~~), nodes~~). Nodes are actors of a behavior, and edges are the behavior relationship

itself. For example, in the case of communication, example actors include processes, which communicate with other processes. The~~-~~ GBM~~-~~ **154** clusters the raw graph based on behaviors of actors and produces a summary (the ~~Polygraph~~polygraph). The ~~Polygraph~~polygraph summarizes behavior at a datacenter level. The GBM also produces "observations" that represent changes detected in the datacenter. Such observations ~~are~~may be based on differences in cumulative behavior (e.g., the baseline) of the datacenter with its current behavior. The~~-~~ GBM~~-~~ **154** can be implemented in any appropriate programming language, such as Java, C, or Golang, using appropriate libraries (as applicable) to handle distributed graph computations (handling large amounts of data analysis in a short amount of time). Apache Spark is another example tool that can be used to compute ~~Polygraphs~~polygraphs. The GBM can also take feedback from users and adjust the model according to that feedback. For example, if a given user is interested in relearning behavior for a particular entity, the GBM can be instructed to "forget" the implicated part of the polygraph.

~~f.~~ GBM ~~Runner~~

~~GBM Runner (~~runner **156**~~)~~ is a microservice that ~~is~~may be responsible for interfacing with GBM **154** and providing GBM **154** with raw graphs (e.g., using a query language, such as SQL, to push any computations it can to ~~database 142~~data store **30**). GBM ~~Runner~~runner **156** may also ~~inserts Polygraph~~insert polygraph output from GBM **154** to ~~database 142.~~ data store **30**. GBM ~~Runner~~ runner **156** can be implemented in any appropriate programming language, such as Java or C, using SQL/JDBC libraries to interact with ~~database 142~~data store **30** to insert and query data.

~~g. EventGen~~

~~EventGen (~~Alert generator **158**~~)~~ is a microservice that ~~is~~may be responsible for generating alerts. ~~It examines~~ Alert generator **158** may examine observations (e.g., produced by GBM **154**) in aggregate, ~~deduplicates~~deduplicate them, and ~~scores~~score them. Alerts ~~are~~may be generated for observations with a score exceeding a threshold. ~~EventGen~~Alert generator **158** may also ~~computes~~compute (or retrieves, as applicable) data that a customer (e.g., ~~Alice~~user A or ~~Bob~~user B) might need when reviewing the alert. Examples of events that can be detected by data platform ~~10~~**212** (and alerted on by ~~EventGen~~alert generator **158**) include, but are not limited to the following:

new user: This event ~~is~~may be created the first time a user (e.g., of node **116**) is first observed by an agent within a datacenter.

user launched new binary: This event ~~is~~may be generated when an interactive user launches an application for the first time.

new privilege escalation: This event ~~is~~may be generated when user privileges are escalated and a new application is run.

new application or container: This event ~~is~~may be generated when an application or container is seen for the first time.

new external connection: This event ~~is~~may be generated when a connection to an external IP/domain is made from a new application.

new external host or IP: This event ~~is~~may be generated when a new external host or IP is involved in a connection with a datacenter.

new internal connection: This event ~~is~~may be generated when a connection between internal-only applications is seen for the first time.

new external client: This event ~~is~~may be generated when a new external connection is seen for an application which typically does not have external connections.

new parent: This event ~~is~~may be generated when an application is launched by a different parent.
connection to known bad IP/domain: ~~Platform **102**~~Data platform **12** maintains (or can otherwise access) one or more reputation feeds. If an environment makes a connection to a known bad IP or domain, an event will be generated.
login from a known bad IP/domain: An event ~~is~~may be generated when a successful connection to a datacenter from a known bad IP is observed by data platform ~~**102**~~**12**.
~~EventGen~~ Alert generator **158** can be implemented in any appropriate programming language, such as Java or C, using SQL/JDBC libraries to interact with ~~database **142**~~data store **30** to insert and query data. In various embodiments, ~~EventGen~~ alert generator **158** also uses one or more machine learning libraries, such as Spark's MLLib (e.g., to compute scoring of various observations). ~~EventGen~~ Alert generator **158** can also take feedback from users about which kinds of events are of interest and which to suppress.
~~h.~~ QsJobServer
~~QsJobServer (~~ **160**~~)~~ is a microservice that ~~looks~~may look at all the data produced by data platform ~~**102**~~**12** for an hour, and ~~compiles~~compile a materialized view (MV) out of the data to make queries faster. The MV helps make sure that the queries customers most frequently run, and data that they search for, can be easily queried and answered. QsJobServer **160** may also ~~precomputes~~precompute and ~~caches~~cache a variety of different metrics so that they can quickly be provided as answers at query time. QsJobServer **160** can be implemented using any appropriate programming language, such as Java or C, using SQL/JDBC libraries. ~~The~~ In some examples, QsJobServer ~~**160**~~ is able to compute an MV efficiently at scale, where there could be a large number of joins. An SQL engine, such as Oracle, can be used to efficiently execute the SQL, as applicable.
~~i. Alert Notifier~~
Alert notifier **162** is a microservice that ~~takes~~may take alerts produced by ~~EventGen~~alert generator **158** and ~~sends the alerts~~send them to customers' integrated Security Information and Event Management (SIEM) products (e.g., Splunk, Slack, etc.). Alert notifier **162** can be implemented using any appropriate programming language, such as Java or C. Alert notifier **162** can be configured to use an email service (e.g., AWS SES or pagerduty) to send emails. Alert notifier **162** may also ~~provides~~provide templating support (e.g., Velocity or Moustache) to manage templates and structured notifications to SIEM products.
~~j. Reporting Module~~
Reporting module **164** is a microservice that may be responsible for creating reports out of customer data (e.g., daily summaries of events, etc.) and providing those reports to customers (e.g., via email). Reporting module **164** can be implemented using any appropriate programming language, such as Java or C. Reporting module **164** can be configured to use an email service (e.g., AWS SES or pagerduty) to send emails. Reporting module **164** may also ~~provides~~provide templating support (e.g., Velocity or Moustache) to manage templates (e.g., for constructing HTML-based email).
~~k. Web Frontend~~
Web app **120** is a microservice that provides a user interface to data collected and processed on data platform ~~**102**~~**12**. Web app **120** ~~provides~~may provide login, authentication, query, data visualization, etc. features. Web app **120** ~~includes~~may, in some embodiments, include both client and server elements. Example ways the server elements can be implemented are using Java DropWizard or Node.Js to serve business logic, and a combination of JSON/HTTP to manage the service. Example ways the client elements can be implemented are using frameworks such as

React, Angular, or Backbone. JSON, jQuery, and JavaScript libraries (e.g., underscore) can also be used.

~~1. Query Service~~

Query service **166** is a microservice that ~~manages~~may manage all database access for web app **120**. Query service **166** abstracts out data obtained from ~~database **142**~~data store **30** and provides a JSON-based REST API service to web app **120**. Query service **166** ~~generates~~may generate SQL queries for the REST APIs that it receives at run time. Query service **166** can be implemented using any appropriate programming language, such as Java or C and SQL/JDBC libraries, or an SQL framework such as jOOQ. Query service **166** can internally make use of a variety of types of databases, including ~~postgres,~~a relational database engine **168** (e.g., AWS Aurora ~~(168)~~) and/or ~~Snowflake (142)~~ data store **30** to manage data for clients. Examples of tables that query service **166** manages are OLTP tables and data warehousing tables.

~~m.~~Cache **170** may be implemented by Redis ~~(170) is an open source project which~~ and/or any other service that provides a key-value store. ~~Platform **102**~~Data platform **12** can use ~~Redis as a~~ cache ~~**170**~~ to keep information for frontend services about users. Examples of such information include valid tokens for a customer, valid cookies of customers, the last time a customer tried to login, etc.

~~2. Example Processing~~

FIG. ~~18~~ **3**A illustrates an example of a process for detecting anomalies in a network environment. In various embodiments, process ~~**1800**~~**300** is performed by data platform ~~**102**~~**12**. The process begins at ~~**1802**~~**301** when data associated with activities occurring in a network environment (such as ~~ACME's~~entity A's datacenter) is received. One example of such data that can be received at ~~**1802**~~**301** is agent-collected data described above (e.g., in conjunction with process **200**).

At ~~**1804**~~**302**, a logical graph model is generated, using at least a portion of the monitored activities. A variety of approaches can be used to generate such logical graph models, and a variety of logical graphs can be generated (whether using the same, or different approaches). The following is one example of how data received at ~~**1802**~~**301** can be used to generate and maintain a model.

During bootstrap, data platform ~~**102**~~**12** creates an aggregate graph of physical connections (also referred to herein as an aggregated physical graph) by matching connections that occurred in the first hour into communication pairs. Clustering is then performed on the communication pairs. Examples of such clustering, described in more detail below, include performing Matching Neighbor clustering and similarity (e.g., SimRank) clustering. Additional processing can also be performed (and is described in more detail below), such as by splitting clusters based on application type, and annotating nodes with DNS query information. The resulting graph (also referred to herein as a base graph or common graph) can be used to generate a variety of models, where a subset of node and edge types (described in more detail below) and their properties are considered in a given model. One example of a model is a UID to UID model (also referred to herein as a Uid2Uid model) which clusters together processes that share a username and show similar privilege change behavior. Another example of a model is a CType model, which clusters together processes that share command line similarity. Yet another example of a model is a PType model, which clusters together processes that share behaviors over time.

Each hour (or any other predetermined time interval) after bootstrap, a new snapshot is taken (i.e., data collected about a datacenter in the last hour is processed) and information from the new snapshot is merged with existing data to create and (as additional data is

collected/processed) maintain a cumulative graph. The cumulative graph (also referred to herein as a cumulative PType graph and a polygraph) is a running model of how processes behave over time. Nodes in the cumulative graph are PType nodes, and provide information such as a list of all active processes and PIDs in the last hour, the number of historic total processes, the average number of active processes per hour, the application type of the process (e.g., the CType of the PType), and historic CType information/frequency. Edges in the cumulative graph can represent connectivity and provide information such as connectivity frequency. The edges can be weighted (e.g., based on number of connections, number of bytes exchanged, etc.). Edges in the cumulative graph (and snapshots) can also represent transitions.

One approach to merging a snapshot of the activity of the last hour into a cumulative graph is as follows. An aggregate graph of physical connections is made for the connections included in the snapshot (as was previously done for the original snapshot used during bootstrap). And, clustering/splitting is similarly performed on the snapshot's aggregate graph. Next, PType clusters in the snapshot's graph are compared against PType clusters in the cumulative graph to identify commonality.

One approach to determining commonality is, for any two nodes that are members of a given CmdType (described in more detail below), comparing internal neighbors and calculating a set membership Jaccard distance. The pairs of nodes are then ordered by decreasing similarity (i.e., with the most similar sets first). For nodes with a threshold amount of commonality (e.g., at least 66% members in common), any new nodes (i.e., appearing in the snapshot's graph but not the cumulative graph) are assigned the same PType identifier as is assigned to the corresponding node in the cumulative graph. For each node that is not classified (i.e., has not been assigned a PType identifier), a network signature is generated (i.e., indicative of the kinds of network connections the node makes, who the node communicates with, etc.). The following processing is then performed until convergence. If a match of the network signature is found in the cumulative graph, the unclassified node is assigned the PType identifier of the corresponding node in the cumulative graph. Any nodes which remain unclassified after convergence are new PTypes and are assigned new identifiers and added to the cumulative graph as new. As applicable, the detection of a new PType can be used to generate an alert. If the new PType has a new CmdType, a severity of the alert can be increased. If any surviving nodes (i.e., present in both the cumulative graph and the snapshot graph) change PTypes, such change is noted as a transition, and an alert can be generated. Further, if a surviving node changes PType and also changes CmdType, a severity of the alert can be increased.

Changes to the cumulative graph (e.g., a new PType or a new edge between two PTypes) can be used (e.g., at ~~1806~~303) to detect anomalies (described in more detail below). Two example kinds of anomalies that can be detected by data platform ~~102~~12 include security anomalies (e.g., a user or process behaving in an unexpected manner) and devops/root cause anomalies (e.g., network congestion, application failure, etc.). Detected anomalies can be recorded and surfaced (e.g., to administrators, auditors, etc.), such as through alerts which are generated at ~~1808~~304 based on anomaly detection.

Additional detail regarding processing performed, by various components depicted in FIG. ~~1~~ 1D (whether performed individually or in combination), in conjunction with model/polygraph construction (e.g., as performed at ~~1804~~302) are provided below.

~~a. Matching Neighbor Clustering~~

As explained above, an aggregated physical graph can be generated on a per customer basis periodically (e.g., once an hour) from raw physical graph information, by matching connections

(e.g., between two processes on two virtual machines). In various embodiments, a deterministic fixed approach is used to cluster nodes in the aggregated physical graph (e.g., representing processes and their communications). As one example, Matching Neighbors Clustering (MNC) can be performed on the aggregated physical graph to determine which entities exhibit identical behavior and cluster such entities together.

FIG. ~~19A~~ 3B depicts a set of example processes (p1, p2, p3, and p4) communicating with other processes (p10 and p11). FIG. ~~19A~~ 3B is a graphical representation of a small portion of an aggregated physical graph showing (for a given time period, such as an hour) which processes in a datacenter communicate with which other processes. Using MNC, processes p1, p2, and p3 will be clustered together (~~1902~~305), as they exhibit identical behavior (they communicate with p10 and only p10). Process p4, which communicates with both p10 and p11, will be clustered separately.

~~b. SimRank~~

In MNC, only those processes exhibiting identical (communication) behavior will be clustered. In various embodiments, an alternate clustering approach can also/instead be used, which uses a similarity measure (e.g., constrained by a threshold value, such as a 60% similarity) to cluster items. In some embodiments, the output of MNC is used as input to SimRank, in other embodiments, MNC is omitted.

FIG. ~~19B~~ 3C depicts a set of example processes (p4, p5, p6) communicating with other processes (p7, p8, p9). As illustrated, most of nodes p4, p5, and p6 communicate with most of nodes p7, p8, and p9 (as indicated in FIG. ~~19B~~ 3C with solid connection lines). As one example, process p4 communicates with process p7 (~~1952~~310), process p8 (~~1954~~311), and process p9 (~~1956~~312). An exception is process p6, which communicates with processes p7 and p8, but does not communicate with process p9 (as indicated by dashed line ~~1958~~313). If MNC were applied to the nodes depicted in FIG. ~~19B~~ 3C, nodes p4 and p5 would be clustered (and node p6 would not be included in their cluster).

One approach to similarity clustering is to use SimRank. In an embodiment of the SimRank approach, for a given node v in a directed graph, I(v) and O(v) denote the respective set of in-neighbors and out-neighbors of v. Individual in-neighbors are denoted as $I_i(v)$, for $1 \le i \le |I(v)|$, and individual out-neighbors are denoted as $O_i(v)$, for $1 \le i \le |O(v)|$. The similarity between two objects a and b can be denoted by $s(a,b) \in [1,0]$. A recursive equation (hereinafter "the SimRank equation") can be written for s(a,b), where, if a=b, then s(a,b) is defined as 1, otherwise,

s (a,b~~)=c□)~~ -

C|I (a~~)□ □)|~~ |I (b~~)□ ∑)|=~~∑i=1□|I (a~~)□ ∑)|~~ ∑j=1□|I (b~~)□)|~~ s (Ii~~((~~a),Ij~~((~~b))

where C is a constant between 0 and 1. One example value for the decay factor C is 0.8 (and a fixed number of iterations such as five). Another example value for the decay factor C is 0.6 (and/or a different number of iterations). In the event that a or b has no in-neighbors, similarity is set to s(a,b)=0, so the summation is defined to be 0 when I(a)=Ø or I(b)=Ø.

The SimRank equations for a graph G can be solved by iteration to a fixed point. Suppose n is the number of nodes in G. For each iteration k, n^2 entries $s_k(*,*)$ are kept, where $s_k(a,b)$ gives the score between a and b on iteration k. Successive computations of $s_{k+1}(*,*)$ are made based on $s_k(*,*)$. Starting with $s_0(*,*)$, where each $s_0(a,b)$ is a lower bound on the actual SimRank score s(a,b):

~~SimRank score s (a,b):~~s0 ~~((~~a,b)={1,if~~ ~~a=b,0,if~~ ~~a≠b.

The SimRank equation can be used to compute $s_{k+1}(a, b)$ from $s_k(*,*)$ with

sk+1——((a,b)=ClI(a,b)=e□)||I (b)| ∑i=1|I (a)□ □)| ∑j=1|I (b)□ ∑i=1□I (a)□ ∑j=1□I (b)□)| sk——((Ii——((a),Ij——((b))——))
for—— a≠b, and- sk+1——((a, b)=1—— for—— a=b.

On each iteration k+1, the similarity of (a,b) is updated using the similarity scores of the neighbors of (a,b) from the previous iteration k according to the SimRank equation. The values $s_k(*,*)$ are nondecreasing as k increases.

Returning to FIG. ~~19B~~ 3C, while MNC would cluster nodes p4 and p5 together (and not include node p6 in their cluster), application of SimRank would cluster nodes p4-p6 into one cluster (~~1960~~314) and also cluster nodes p7-p9 into another cluster (~~1962~~315).

FIG. ~~19C~~ 3D depicts a set of processes, and in particular server processes s1 and s2, and client processes c1, c2, c3, c4, c5, and c6. Suppose only nodes s1, s2, c1, and c2 are present in the graph depicted in FIG. ~~19C~~ 3D (and the other nodes depicted are omitted from consideration). Using MNC, nodes s1 and s2 would be clustered together, as would nodes c1 and c2. Performing SimRank clustering as described above would also result in those two clusters (s1 and s2, and c1 and c2). As previously mentioned, in MNC, identical behavior is required. Thus, if node c3 were now also present in the graph, MNC would not include c3 in a cluster with c2 and c1 because node c3 only communicates with node s2 and not node s1. In contrast, a SimRank clustering of a graph that includes nodes s1, s2, c1, c2, and c3 would result (based, e.g., on an applicable selected decay value and number of iterations) in a first cluster comprising nodes s1 and s2, and a second cluster of c1, c2, and c3. As an increasing number of nodes which communicate with server process s2, and do not also communicate with server process s1, are included in the graph (e.g., as c4, c5, and c6 are added), under SimRank, nodes s1 and s2 will become decreasingly similar (i.e., their intersection is reduced).

In various embodiments, SimRank is modified (from what is described above) to accommodate differences between the asymmetry of client and server connections. As one example, SimRank can be modified to use different thresholds for client communications (e.g., an 80% match among nodes c1-c6) and for server communications (e.g., a 60% match among nodes s1 and s2). Such modification can also help achieve convergence in situations such as where a server process dies on one node and restarts on another node.

~~c. Cluster Splitting (e.g., Based on Application)~~

The application of MNC/SimRank to an aggregated physical graph results in a smaller graph, in which processes which are determined to be sufficiently similar are clustered together. Typically, clusters generated as output of MNC will be underinclusive. For example, for the nodes depicted in FIG. ~~19B~~ 3C, process p6 will not be included in a cluster with processes p4 and p5, despite substantial similarity in their communication behaviors. The application of SimRank (e.g., to the output of MNC) helps mitigate the underinclusiveness of MNC, but can result in overly inclusive clusters. As one example, suppose (returning to the nodes depicted in FIG. ~~19A~~ 3B) that as a result of applying SimRank to the depicted nodes, nodes p1-p4 are all included in a single cluster. Both MNC and SimRank operate agnostically of which application a given process belongs to. Suppose processes p1-p3 each correspond to a first application (e.g., an update engine), and process p4 corresponds to a second application (e.g., sshd). Further suppose process p10 corresponds to contact with AWS. Clustering all four of the processes together (e.g., as a result of SimRank) could be problematic, particularly in a security context (e.g., where granular information useful in detecting threats would be lost).

As previously mentioned, data platform ~~102 maintains~~12 may maintain a mapping between processes and the applications to which they belong. In various embodiments, the output of

SimRank (e.g., SimRank clusters) is split based on the applications to which cluster members belong (such a split is also referred to herein as a "CmdType split"). If all cluster members share a common application, the cluster remains. If different cluster members originate from different applications, the cluster members are split along application-type (CmdType) lines. Using the nodes depicted in FIG. ~~19C~~ 3D as an example, suppose that nodes c1, c2, c3, and c5 all share "update engine" as the type of application to which they belong (sharing a CmdType). Suppose that node c4 belongs to "ssh," and suppose that node c6 belongs to "bash." As a result of SimRank, all six nodes (c1-c6) might be clustered into a single cluster. After a CmdType split is performed on the cluster, however, the single cluster will be broken into three clusters (c1, c2, c3, c5; c4; and c6). Specifically, the resulting clusters comprise processes associated with the same type of application, which exhibit similar behaviors (e.g., communication behaviors). Each of the three clusters resulting from the CmdType split represents, respectively, a node (also referred to herein as a PType) of a particular CmdType. Each PType is given a persistent identifier and stored persistently as a cumulative graph.

A variety of approaches can be used to determine a CmdType for a given process. As one example, for some applications (e.g., sshd), a one-to-one mapping exists between the CmdType and the application/binary name. Thus, processes corresponding to the execution of sshd will be classified using a CmdType of sshd. In various embodiments, a list of common application/binary names (e.g., sshd, apache, etc.) is maintained by data platform ~~102~~12 and manually curated as applicable. Other types of applications (e.g., Java, Python, and Ruby) are multi-homed, meaning that several very different applications may all execute using the binary name, "java." For these types of applications, information such as command line/execution path information can be used in determining a CmdType. In particular, the subapplication can be used as the CmdType of the application, and/or term frequency analysis (e.g., TF/IDF) can be used on command line information to group, for example, any marathon related applications together (e.g., as a python.marathon CmdType) and separately from other Python applications (e.g., as a python.airflow CmdType).

In various embodiments, machine learning techniques are used to determine a CmdType. The CmdType model is constrained such that the execution path for each CmdType is unique. One example approach to making a CmdType model is a random forest based approach. An initial CmdType model is bootstrapped using process parameters (e.g., available within one minute of process startup) obtained using one hour of information for a given customer (e.g., ~~ACME~~entity A). Examples of such parameters include the command line of the process, the command line of the process's parent(s) (if applicable), the uptime of the process, UID/EUID and any change information, TTY and any change information, listening ports, and children (if any). Another approach is to perform term frequency clustering over command line information to convert command lines into cluster identifiers.

The random forest model can be used (e.g., in subsequent hours) to predict a CmdType for a process (e.g., based on features of the process). If a match is found, the process can be assigned the matching CmdType. If a match is not found, a comparison between features of the process and its nearest CmdType (e.g., as determined using a Levenstein distance) can be performed. The existing CmdType can be expanded to include the process, or, as applicable, a new CmdType can be created (and other actions taken, such as generating an alert). Another approach to handling processes which do not match an existing CmdType is to designate such processes as unclassified, and once an hour, create a new random forest seeded with process information from a sampling of classified processes (e.g., 10 or 100 processes per CmdType) and the new

processes. If a given new process winds up in an existing set, the process is given the corresponding CmdType. If a new cluster is created, a new CmdType can be created.

~~d. Graph Behavior Model (GBM)~~

~~i. GBM Overview~~

~~Concept~~

Conceptually, a polygraph represents the smallest possible graph of clusters that preserve a set of rules (e.g., in which nodes included in the cluster must share a CmdType and behavior). As a result of performing MNC, SimRank, and cluster splitting (e.g., CmdType splitting) many processes are clustered together based on commonality of behavior (e.g., communication behavior) and commonality of application type. Such clustering represents a significant reduction in graph size (e.g., compared to the original raw physical graph). Nonetheless, further clustering can be performed (e.g., by iterating on the graph data using the GBM to achieve such a polygraph). As more information within the graph is correlated, more nodes can be clustered together, reducing the size of the graph, until convergence is reached and no further clustering is possible.

FIG. ~~19D~~ 3E depicts two pairs of clusters. In particular, cluster ~~1964~~320 represents a set of client processes sharing the same CmdType ("a1"), communicating (collectively) with a server process having a CmdType ("a2"). Cluster ~~1968~~322 also represents a set of client processes having a CmdType a1 communicating with a server process having a CmdType a2. The nodes in clusters ~~1964~~320 and ~~1968~~322 (and similarly nodes in ~~1966~~321 and ~~1970~~323) remain separately clustered (as depicted) after MNC/SimRank/CmdType splitting—isolated islands. One reason this could occur is where server process ~~1966~~321 corresponds to processes executing on a first machine (having an IP address of 1.1.1.1). The machine fails and a new server process ~~1970~~323 starts, on a second machine (having an IP address of 2.2.2.2) and takes over for process ~~1966~~321.

Communications between a cluster of nodes (e.g., nodes ~~1964~~ of cluster **320**) and the first IP address can be considered different behavior from communications between the same set of nodes and the second IP address, and thus communications ~~1972~~324 and ~~1974~~325 will not be combined by MNC/SimRank in various embodiments. Nonetheless, it could be desirable for nodes ~~1964/1968~~ of clusters **320**/**322** to be combined (into cluster ~~1976~~326), and for nodes ~~1966/1970~~ of clusters **321**/**323** to be combined (into cluster ~~1978~~327), as representing (collectively) communications between a1 and a2. One task that can be performed by data platform ~~10~~12 is to use DNS query information to map IP addresses to logical entities. As will be described in more detail below, GBM **154** can make use of the DNS query information to determine that graph nodes ~~1964~~ of cluster **320** and graph nodes ~~1968~~ of cluster **322** both made DNS queries for "appserverabc.example.com," which first resolved to 1.1.1.1 and then to 2.2.2.2, and to combine nodes ~~1964/1968~~320/322 and ~~1966/1970~~321/323 together into a single pair of nodes (~~1976~~326 communicating with ~~1978~~327).

In various embodiments, GBM **154** operates in a batch manner in which it receives as input the nodes and edges of a graph for a particular time period along with its previous state, and generates as output clustered nodes, cluster membership edges, cluster-to-cluster edges, events, and its next state.

GBM **154** ~~does~~may not try to consider all types of entities and their relationships that may be available in a conceptual common graph all at once. Instead, GBM uses a concept of models where a subset of node and edge types and their properties are considered in a given model. Such an approach is helpful for scalability, and also to help preserve detailed information (of particular

importance in a security context)—as clustering entities in a more complex and larger graph could result in less useful results. In particular, such an approach allows for different types of relationships between entities to be preserved/more easily analyzed.

While GBM **154** can be used with different models corresponding to different subgraphs, core abstractions remain the same across types of models~~:~~.

~~Each~~For example, each node type in a GBM model is considered to belong to a class. The class can be thought of as a way for the GBM to split nodes based on the criteria it uses for the model. The class for a node is represented as a string whose value is derived from the node's key and properties depending on the GBM Model. Note that different GBM models may create different class values for the same node. For each node type in a given GBM model, GBM **154** can generate clusters of nodes for that type. A GBM generated cluster for a given member node type cannot span more than one class for that node type. GBM **154** generates edges between clusters that have the same types as the edges between source and destination cluster node types.

~~The~~Additionally or alternatively, the processes described herein as being used for a particular model can be used (can be the same) across models, and different models can also be configured with different settings.

~~The~~Additionally or alternatively, the node types and the edge types may correspond to existing types in the common graph node and edge tables but this is not necessary. Even when there is a correspondence, the properties provided to GBM **154** are not limited to the properties that are stored in the corresponding graph table entries. They can be enriched with additional information before being passed to GBM **154**.

~~Graph Input~~

Logically, the input for a GBM model can be characterized in a manner that is similar to other graphs. Edge triplets can be expressed, for example, as an array of source node type, edge type, and destination node type. And, each node type is associated with node properties, and each edge type is associated with edge properties. Other edge triplets can also be used (and/or edge triplets can be extended) in accordance with various embodiments.

Note that the physical input to the GBM model need not (and does not, in various embodiments) conform to the logical input. For example, the edges in the PtypeConn model correspond to edges between Matching Neighbors (MN) clusters, where each process node has an MN cluster identifier property. In the User ID to User ID model (also referred to herein as the Uid2Uid model), edges are not explicitly provided separately from nodes (as the euid array in the node properties serves the same purpose). In both cases, however, the physical information provides the applicable information necessary for the logical input.

~~State Input~~

The state input for a particular GBM model can be stored in a file, a database, or other appropriate storage. The state file (from a previous run) is provided, along with graph data, except for when the first run for a given model is performed, or the model is reset. In some cases, no data may be available for a particular model in a given time period, and GBM may not be run for that time period. As data becomes available at a future time, GBM can run using the latest state file as input.

~~Graph Output~~

GBM **154** outputs cluster nodes, cluster membership edges, and inter-cluster relationship edges that are stored (in some embodiments) in the graph node tables: node_c, node_cm, and node_icr, respectively. The type names of nodes and edges may conform to the following rules:

-

- A given node type can be used in multiple different GBM models. The type names of the cluster nodes generated by two such models for that node type will be different. For instance, process type nodes will appear in both PtypeConn and Uid2Uid models, but their cluster nodes will have different type names.
- The membership edge type name is "MemberOf."
- The edge type names for cluster-to-cluster edges will be the same as the edge type names in the underlying node-to-node edges in the input.

~~Event Types~~

The following are example events GBM **154** can generate:

new class

, new cluster

, new edge from class to class

, split class (the notion that GBM **154** considers all nodes of a given type and class to be in the same cluster initially and if GBM **154** splits them into multiple clusters, it is splitting a class~~)~~), new edge from cluster and class, new edge between cluster and cluster, and/or new edge from class to cluster.

~~new edge from cluster and class~~

~~new edge between cluster and cluster~~

~~new edge from class to cluster~~

One underlying node or edge in the logical input can cause multiple types of events to be generated. Conversely, one event can correspond to multiple nodes or edges in the input. Not every model generates every event type.

~~ii. GBM Models~~

Additional information regarding examples of data structures/models that can be used in conjunction with models used by data platform ~~102~~**12** is now provided ~~in this section~~.

In some examples, a PTypeConn Model

~~This model~~ clusters nodes of the same class that have similar connectivity relationships. For example, if two processes had similar incoming neighbors of the same class and outgoing neighbors of the same class, they could be clustered.

The node input to the PTypeConn model for a given time period includes non-interactive (i.e., not associated with tty) process nodes that had connections in the time period and the base graph nodes of other types (IP Service Endpoint (IPSep) comprising an IP address and a port), DNS Service Endpoint (DNSSep) and IPAddress) that have been involved in those connections. The base relationship is the connectivity relationship for the following type triplets:

-
  - Process, ConnectedTo, Process
  - Process, ConnectedTo, IP Service Endpoint (IPSep)
  - Process, ConnectedTo, DNS Service Endpoint (DNSSep)
  - IPAddress, ConnectedTo, ProcessProcess, DNS, ConnectedTo, Process

The edge inputs to this model are the ConnectedTo edges from the MN cluster, instead of individual node-to-node ConnectedTo edges from the base graph. The membership edges created by this model refer to the base graph node type provided in the input.
Class Values:
The class values of nodes are determined as follows depending on the node type (e.g., Process nodes, IPSep nodes, DNSSep nodes, and IP Address nodes).
Process Nodes:

-
  - if exe_path contains java then “java <cmdline_term_1> . . . ”
  - else if exe_path contains python then “python <cmdline_term_1> . . . ”
  - else “last_part_of_exe_path”

IPSep Nodes:

~~IPSep Nodes:~~

  - if IP_internal then “IntIPS”
  - else if severity=0 then “<IP_addr~~>:<~~>: <protocol~~>:<~~>: <port>”

else “<IP_addr~~>:<~~>: <port~~>_~~>BadIP”

-

DNSSep Nodes:

  - if IP_internal=1 then “<hostname>”
  - else if severity=0 then “<hostname~~>:<~~>: <protocol>: port”

else “<hostname~~>:<~~>: <port~~>_~~>BadIP”

-

IPAddress ~~nodes (will appear only~~Nodes (Will Appear Only on ~~client side~~Client Side):

  - if IP_internal=1 then “IPIntC”
  - else if severity=0 then “ExtIPC”

else “ExtBadIPC”

-

Events:

A new class event in this model for a process node is equivalent to seeing a new CType being involved in a connection for the first time. Note that this does not mean the CType was not seen before. It is possible that it was previously seen but did not make a connection at that time.
A new class event in this model for an IPSep node with IP_internal=0 is equivalent to seeing a connection to a new external IP address for the first time.
A new class event in this model for a DNSSep node is equivalent to seeing a connection to a new domain for the first time.

A new class event in this model for an ~~IPAddress~~IP Address node with IP_internal=0 and severity=0 is equivalent to seeing a connection from any external IP address for the first time.
A new class event in this model for an IPAddress node with IP_internal=0 and severity >0 is equivalent to seeing a connection from any bad external IP address for the first time.
A new class to class to edge from a class for a process node to a class for a process node is equivalent to seeing a communication from the source CType making a connection to the destination CType for the first time.
A new class to class to edge from a class for a process node to a class for a DNSSep node is equivalent to seeing a communication from the source CType making a connection to the destination domain name for the first time.

An IntPConn Model

~~This model is~~ may be similar to the PtypeConn Model, except that connection edges between parent/child processes and connections between processes where both sides are not interactive are filtered out.

A Uid2Uid Model

~~This model clusters~~ may cluster processes with the same username that show similar privilege change behavior. For instance, if two processes with the same username had similar effective user values, launched processes with similar usernames, and were launched by processes with similar usernames, then they could be clustered.
An edge between a source cluster and destination cluster generated by this model means that all of the processes in the source cluster had a privilege change relationship to at least one process in the destination cluster.
The node input to this model for a given time period includes process nodes that are running in that period. The value of a class of process nodes is "<username>".
The base relationship that is used for clustering is privilege change, either by the process changing its effective user ID, or by launching a child process which runs with a different user.
The physical input for this model includes process nodes (only), with the caveat that the complete ancestor hierarchy of process nodes active (i.e., running) for a given time period is provided as input even if an ancestor is not active in that time period. Note that effective user IDs of a process are represented as an array in the process node properties, and launch relationships are available from ppid_hash fields in the properties as well.
A new class event in this model is equivalent to seeing a user for the first time.
A new class to class edge event is equivalent to seeing the source user making a privilege change to the destination user for the first time.

A Ct2Ct Model

~~This model clusters~~ may cluster processes with the same CType that show similar launch behavior. For instance, if two processes with the same CType have launched processes with similar CTypes, then they could be clustered.
The node input to this model for a given time period includes process nodes that are running in that period. The value class of process nodes is CType (similar to how it is created for the PtypeConn Model).
The base relationship that is used for clustering is a parent process with a given CType launching a child process with another given destination CType.
The physical input for this model includes process nodes (only) with the caveat that the complete ancestor hierarchy active process nodes (i.e., that are running) for a given time period is provided

as input even if an ancestor is not active in that time period. Note that launch relationships are available from ppid_hash fields in the process node properties.
An edge between a source cluster and destination cluster generated by this model means that all of the processes in the source cluster launched at least one process in the destination cluster.
A new class event in this model is equivalent to seeing a CType for the first time. Note that the same type of event will be generated by the PtypeConn Model as well.
A new class to class edge event is equivalent to seeing the source CType launching the destination CType for the first time.
An MTypeConn Model
~~This model clusters~~ may cluster nodes of the same class that have similar connectivity relationships. For example, if two machines had similar incoming neighbors of the same class and outgoing neighbors of the same class, they could be clustered.
A new class event in this model will be generated for external IP addresses or (as applicable) domain names seen for the first time. Note that a new class to class to edge Machine, class to class for an IPSep or DNSName node will also be generated at the same time.
The membership edges generated by this model will refer to Machine, ~~IPAddress~~IP Address, DNSName, and IPSep nodes in the base graph. Though the nodes provided to this model are IPAddress nodes instead of IPSep nodes, the membership edges it generates will refer to IPSep type nodes. Alternatively, the base graph can generate edges between Machine and IPSep node types. Note that the Machine to IPAddress edges have tcp_dst_ports/udp_dst_ports properties that can be used for this purpose.
The node input to this model for a given time period includes machine nodes that had connections in the time period and the base graph nodes of other types (IPAddress and DNSName) that were involved in those connections.
The base relationship is the connectivity relationship for the following type triplets:

-
  - Machine, ConnectedTo, Machine
  - Machine, ConnectedTo, IPAddress
  - Machine, ConnectedTo, DNSName
  - IPAddress, ConnectedTo, Machine, DNS, ConnectedTo, Machine

The edge inputs to this model are the corresponding ConnectedTo edges in the base graph.
Class Values:
Machine:
The class value for all Machine nodes is “Machine.”
The machine_terms property in the Machine nodes is used, in various embodiments, for labeling machines that are clustered together. If a majority of the machines clustered together share a term in the machine_terms, that term can be used for labeling the cluster.
IPSep:
The class value for IPSep nodes is determined as follows:

-
  - if IP_internal then “IntIPS”

- else
- if severity=0 then “<ip_addr>:<>: <protocol>:<>: <port>”

else “<IP_addr_BadIP>”

- IPAddress:

The class value for IpAddress nodes is determined as follows:

-
  - if IP_internal then “IntIPC”
  - else
  - if severity=0 then “ExtIPC”

else “ExtBadIPC”

- DNSName:

The class value for DNSName nodes is determined as follows:

-
  - if severity=0 then “<hostname>”
  - else then “<hostname>_>BadIP”

~~iii. GBM Event Types~~

An example structure for a New Class Event is now described.

~~Structure:~~

The key field for this event type looks as follows (using the PtypeConn model as an example):

-
  - {
  - “node”: {
  -
    - “class”: {
    - “cid”: “httpd”
    - },
    - “key”: {
    - “cid”: “29654”
    - },

- “type”: “PtypeConn”
- }
- }

It contains the class value and also the ID of the cluster where that class value is observed. Multiple clusters can be observed with the same value in a given time period. It contains the class value and also the ID of the cluster where that class value is observed. Multiple clusters can be observed with the same value in a given time period. Accordingly, in some embodiments, GBM **154** generates multiple events of this type for the same class value.
The properties field looks as follows:

-
- {
- “set_size”: 5
- }

The set_size indicates the size of the cluster referenced in the keys field.
Conditions:
For a given model and time period, multiple NewClass events can be generated if there is more than one cluster in that class. NewNode events will not be generated separately in this case.
Example New Class to Class Edge Event
Structure:
The key field for this event type looks as follows (using the PtypeConn model as an example):

-
- “edge”: {
-
- “dst_node”: {
- “class”: {
- “cid”: “java war”
- },
- “key”: {
- “cid”: “27635”
- },
- “type”: “PtypeConn”
- },
- “src_node”: {

- “class”: {
- “cid”: “IntIPC”
- },
- “key”: {
- “cid”: “20881”
- },
- “type”: “PtypeConn”
- },
- “type”: “ConnectedTo”
- }
- }

The key field contains source and destination class values and also source and destination cluster identifiers (i.e., the src/dst_node:key.cid represents the src/dst cluster identifier).
In a given time period for a given model, an event of this type could involve multiple edges between different cluster pairs that have the same source and destination class values. GBM **154** can generate multiple events in this case with different source and destination cluster identifiers.
The props fields look as follows for this event type:

-
- {
- “dst_set_size”: 2,
- “src_set_size”: 1
- }

The source and destination sizes represent the sizes of the clusters given in the keys field.
Conditions:
For a given model and time period, multiple NewClassToClass events can be generated if there are more than one pair of clusters in that class pair. NewNodeToNode events are not generated separately in this case.
~~iv.~~ Combining Events at the Class Level
~~For~~: for a given model and time period, the following example types of events can represent multiple changes in the underlying GBM cluster level graph in terms of multiple new clusters or multiple new edges between clusters:

-
- NewClass

- NewEdgeClassToClass
- NewEdgeNodeToClass
- NewEdgeClassToNode

Multiple NewClass events with the same model and class can be output if there are multiple clusters in that new class.
Multiple NewEdgeClassToClass events with the same model and class pair can be output if there are multiple new cluster edges within that class pair.
Multiple NewEdgeNodeToClass events with the same model and destination class can be output if there are multiple new edges from the source cluster to the destination clusters in that destination class (the first time seeing this class as a destination cluster class for the source cluster).
Multiple NewEdgeClassToNode events with the same model and source class can be output if there are multiple new edges from source clusters to the destination clusters in that source class (the first time seeing this class as a source cluster class for the destination cluster).
These events may be combined at the class level and treated as a single event when it is desirable to view changes at the class level, e.g., when one wants to know when there is a new CType.
~~Also note that~~In some examples, different models may have partial overlap in the types of nodes they use from the base graph. Therefore, they can generate NewClass type events for the same class. NewClass events can also be combined across models when it is desirable to view changes at the class level.

~~III. Extended User Session Tracking~~

Using techniques herein, actions can be associated with processes and (e.g., by associating processes with users) actions can thus also be associated with extended user sessions. Such information can be used to track user behavior correctly, even where a malicious user attempts to hide his trail by changing user identities (e.g., through lateral movement). Extended user session tracking can also be useful in operational use cases without malicious intent, e.g., where users make original logins with distinct usernames (e.g., “charlie” or “dave”) but then perform actions under a common username (e.g., “admin” or “support”). One such example is where multiple users with administrator privileges exist, and they need to gain superuser privilege to perform a particular type of maintenance. It may be desirable to know which operations are performed (as the superuser) by which original user when debugging issues. In the following examples describing extended user session tracking, reference is generally made to using the secure shell (ssh) protocol as implemented by openssh (on the server side) as the mechanism for logins. However, extended user session tracking is not limited to the ssh protocol or a particular limitation and the techniques described herein can be extended to other login mechanisms.
On any given machine, there will be a process that listens for and accepts ssh connections on a given port. This process can run the openssh server program running in daemon mode or it could be running another program (e.g., initd on a Linux system). In either case, a new process running openssh will be created for every new ssh login session and this process can be used to identify an ssh session on that machine. This process is called the “privileged” process in openssh.
After authentication of the ssh session, when an ssh client requests a shell or any other program to be run under that ssh session, a new process that runs that program will be created under (i.e., as a child of) the associated privileged process. If an ssh client requests port forwarding to be performed, the connections will be associated with the privileged process.

In modern operating systems such as Linux and Windows, each process has a parent process (except for the very first process) and when a new process is created the parent process is known. By tracking the parent-child hierarchy of processes, one can determine if a particular process is a descendant of a privileged openssh process and thus if it is associated with an ssh login session. For user session tracking across machines (or on a single machine with multiple logins) in a distributed environment, it is established when two login sessions have a parent-child relationship. After that, the "original" login session, if any, for any given login session can be determined by following the parent relationship recursively.

FIG. ~~20~~ 3F is a representation of a user logging into a first machine and then into a second machine from the first machine, as well as information associated with such actions. In the example of FIG. ~~20~~ 3F, a user, Charlie, logs into Machine A (~~2002~~331) from a first IP address (~~2004~~332). As part of the login process, he provides a username (~~2006~~333). Once connected to Machine A, an openssh privileged process (~~2008~~334) is created to handle the connection for the user, and a terminal session is created and a bash process (~~2010~~335) is created as a child. Charlie launches an ssh client (~~2012~~336) from the shell, and uses it to connect (~~2014~~337) to Machine B (~~2016~~338). As with the connection he makes to Machine A, Charlie's connection to Machine B will have an associated incoming IP address (~~2018~~339), in this case, the IP address of Machine A. And, as part of the login process with Machine B, Charlie will provide a username (~~2020~~340) which need not be the same as username ~~2006~~333. An openssh privileged process (~~2022~~341) is created to handle the connection, and a terminal session and child bash process (~~2024~~342) will be created. From the command line of Machine B, Charlie launches a curl command (~~2026~~343), which opens an HTTP connection (**2028**) to an external Machine C (~~2030~~345).

FIG. ~~21~~ 3G is an alternate representation of actions occurring in FIG. ~~20~~ 3F, where events occurring on Machine A are indicated along line ~~2102~~350, and events occurring on Machine B are indicated along line ~~2104~~351. As shown in FIG. ~~21~~ 3G, an incoming ssh connection is received at Machine A (~~2106~~352). Charlie logs in (as user "x") and an ssh privileged process is created to handle Charlie's connection (~~2108~~353). A terminal session is created and a bash process is created (~~2110~~354) as a child of process ~~2108~~353. Charlie wants to ssh to Machine B, and so executes an ssh client on Machine A (~~2112~~355), providing credentials (as user "y") at ~~2114~~356. Charlie logs into Machine B, and an ssh privileged process is created to handle Charlie's connection (~~2116~~357). A terminal session is created and a bash process is created (~~2118~~358) as a child of process ~~2116~~357. Charlie then executes curl (~~2120~~359) to download content from an external domain (via connection ~~2122~~360).

The external domain could be a malicious domain, or it could be benign. Suppose the external domain is malicious (and, e.g., Charlie has malicious intent). It would be advantageous (e.g., for security reasons) to be able to trace the contact with the external domain back to Machine A, and then back to Charlie's IP address. Using techniques described herein (e.g., by correlating process information collected by various agents), such tracking of Charlie's activities back to his original login (~~2000~~330) can be accomplished. In particular, an extended user session can be tracked that associates Charlie's ssh processes together with a single original login and thus original user.

~~A. Data Model~~

As ~~previously explained~~described herein, software agents (such as agent **112**) may run on machines (such as a machine that implements one of nodes **116**) and detect new connections, processes, and/or logins. As also previously explained, such agents send associated records to data platform ~~10~~12 which includes one or more datastores (e.g., ~~database 142~~data store **30**) for persistently storing such data. Such data can be modeled using logical tables, also persisted in

datastores (e.g., in a relational database that provides an SQL interface), allowing for querying of the data. Other datastores such as graph oriented databases and/or hybrid schemes can also be used.

~~1. Common Identifiers~~

The following identifiers are commonly used in the tables:

-
  - MID
  - PID_hash

An ssh login session can be identified uniquely by an (MID, PID_hash) tuple. The MID is a machine identifier that is unique to each machine, whether physical or virtual, across time and space. Operating systems use numbers called process identifiers (PIDs) to identify processes running at a given time. Over time processes may die and new processes may be started on a machine or the machine itself may restart. The PID is not necessarily unique across time in that the same PID value can be reused for different processes at different times. In order to track process descendants across time, one should therefore account for time as well. In order to be able to identify a process on a machine uniquely across time, another number called a PID_hash is generated for the process. In various embodiments, the PID_hash is generated using a collision-resistant hash function that takes the PID, start time, and (in various embodiments, as applicable) other properties of a process.

~~2. Input Data Model~~

Input data collected by agents comprises the input data model and is represented by the following logical tables:

-
  - connections
  - processes
  - logins

~~a. Connections Table~~

~~The~~A connections table ~~maintains~~may maintain records of TCP/IP connections observed on each machine. Example columns included in a connections table are as follows:

Column Name	Description
MID	Identifier of the machine that the ~~connection was~~ connection was observed on.
start_time	Connection start time.
PID_hash	Identifier of the process that was ~~associated with~~ associated with the connection.
src_IP_addr	Source IP address (the connection was ~~initiated from~~ initiated from this IP address).
src_port	Source port.
dst_IP_addr	Destination IP address (the connection ~~was made to~~

Column Name	Description
	was made to this IP address).
dst_port	Destination port.
Prot	Protocol (TCP or UDP).
Dir	Direction of the connection (incoming or ~~outgoing)~~ outgoing) with respect to this machine.

The source fields (IP address and port) correspond to the side from which the connection was initiated. On the destination side, the agent associates an ssh connection with the privileged ssh process that is created for that connection.
For each connection in the system, there will be two records in the table, assuming that the machines on both sides of the connection capture the connection. These records can be matched based on equality of the tuple (src_IP_addr, src_port, dst_IP_addr, dst_port, Prot) and proximity of the start_time fields (e.g., with a one minute upper threshold between the start_time fields).

~~b. Processes Table~~

~~The~~A processes table maintains records of processes observed on each machine. It ~~has~~may have the following columns:

Column Name	Description
MID	Identifier of the machine that the ~~process was~~ process was observed on.
PID_hash	Identifier of the process.
start_time	Start time of the process.
exe_path	The executable path of the process.
PPID_hash	Identifier of the parent process.

~~c. Logins Table~~

~~The~~A logins table ~~maintains~~may maintain records of logins to ~~the~~ machines. It ~~has~~may have the following columns:

Column Name	Description
MID	Identifier of the machine that the login ~~was~~ was observed on.
sshd_PID_hash	Identifier of the sshd privileged process associated with login.
login_time	Time of login.
login_username	Username used in login.

~~3.~~ Output ~~Data Model~~

~~The output~~ data generated by session tracking is represented with the following logical tables:

-

Deleted Cells
Deleted Cells
Deleted Cells
Deleted Cells
Deleted Cells

- login-local-descendant
- login-connection
- login-lineage

Using data in these tables, it is possible to determine descendant processes of a given ssh login session across the environment (i.e., spanning machines). Conversely, given a process, it is possible to determine if it is an ssh login descendant as well as the original ssh login session for it if so.

~~a. Login-Local-Descendant Table~~

~~The~~A login-local-descendant table maintains the local (i.e., on the same machine) descendant processes of each ssh login session. It ~~has~~may have the following columns:

Column Name	Description
MID	Identifier of the machine that the login ~~was observed on.~~
~~sshd_PID_hash~~	~~Identifier of the sshd privileged process associated with login.~~
~~login_time~~	~~Time of login.~~
~~login_username~~	~~Username used in login.~~

~~b. Login-Connections Table~~

~~The login-connections table maintains the connections associated with ssh logins. It has the following columns:~~

~~Column Name~~	~~Description~~
~~MID~~	~~Identifier of the machine that the process was observed on.~~
~~sshd_PID_hash~~	~~Identifier of the sshd privileged process associated with the login.~~
~~login_time~~	~~Time of login.~~
~~login_username~~	~~The username used in the login.~~
~~src_IP_addr~~	~~Source IP address (connection was initiated from this IP address).~~
~~src_port~~	~~Source port.~~
~~dst_IP_addr~~	~~Destination IP address (connection was made to this IP address).~~
~~dst_port~~	~~Destination port.~~

~~c. Login-Lineage Table~~

~~The login-lineage table maintains the lineage of ssh login sessions. It has the following columns:~~

Deleted Cells
Deleted Cells
Deleted Cells
Deleted Cells

~~Column Name~~	~~Description~~
~~MID~~	~~Identifier of the machine that the ssh login~~ was observed on.
sshd_PID_hash	Identifier of the sshd privileged process associated with ~~the~~ login.
~~parent_MID~~	~~Identifier of the machine that the parent ssh login was observed on.~~
~~parent_sshd_PID_hash~~	~~Identifier of the sshd privileged process associated with the parent~~Time of login.
login_time	
~~origin_MID~~	~~Identifier of the machine that the origin ssh login was observed on.~~
~~origin_sshd_PID_hash~~	~~Identifier of the sshd privileged process associated with the origin~~Username used in login.
login_username	

A login-connections table may maintain the connections associated with ssh logins. It may have the following columns:

Column Name	Description
MID	Identifier of the machine that the process was observed on.
sshd_PID_hash	Identifier of the sshd privileged process associated with the login.
login_time	Time of login.
login_username	The username used in the login.
src_IP_addr	Source IP address (connection was initiated from this IP address).
src_port	Source port.
dst_IP_addr	Destination IP address (connection was made to this IP address).
dst_port	Destination port.

A login-lineage table may maintain the lineage of ssh login sessions. It may have the following columns:

Column Name	Description
MID	Identifier of the machine that the ssh login was observed on.
sshd_PID_hash	Identifier of the sshd privileged process associated with the login.
parent_MID	Identifier of the machine that the parent ssh login was observed on.

Column Name	Description
parent_sshd_PID_hash	Identifier of the sshd privileged process associated with the parent login.
origin_MID	Identifier of the machine that the origin ssh login was observed on.
origin_sshd_PID_hash	Identifier of the sshd privileged process associated with the origin login.

The parent_MID and parent_sshd_PID_hash columns can be null if there is no parent ssh login. In that case, the (MID, sshd_PID_hash) tuple will be the same as the (origin_MID, origin_sshd_PID_hash) tuple.

~~B. Example Processing~~

FIG. ~~22~~ 3H illustrates an example of a process for performing extended user tracking. In various embodiments, process **~~2200~~361** is performed by data platform **~~102~~12**. The process begins at **~~2202~~362** when data associated with activities occurring in a network environment (such as ~~ACME's~~entity A's datacenter) is received. One example of such data that can be received at **~~2202~~362** is agent-collected data described above (e.g., in conjunction with process **200**). At **~~2204~~363**, the received network activity is used to identify user login activity. And, at **~~2206~~364**, a logical graph that links the user login activity to at least one user and at least one process is generated (or updated, as applicable). Additional detail regarding process **~~2200~~361**, and in particular, portions **~~2204~~363** and **~~2206~~364** of process **~~2200~~361** are described in more detail below (e.g., in conjunction with discussion of FIG. ~~24~~ 3J).

FIG. ~~23~~ 3I depicts a representation of a user logging into a first machine, then into a second machine from the first machine, and then making an external connection. The scenario depicted in FIG. ~~23~~ 3I is used to describe an example of processing that can be performed on data collected by agents to generate extended user session tracking information. FIG. ~~23~~ 3I is an alternate depiction of the information shown in FIGS. ~~20~~ 3F and ~~21~~ 3G.

At time t1 (**~~2302~~365**), a first ssh connection is made to Machine A (**~~2304~~366**) from an external source (**~~2306~~367**) by a user having a username of "X." In the following example, suppose the external source has an IP address of 1.1.1.10 and uses source port **10000** to connect to Machine A (which has an IP address of 2.2.2.20 and a destination port **22**). External source **~~2306~~367** is considered an external source because its IP address is outside of the environment being monitored (e.g., is a node outside of ~~ACME's~~entity A's datacenter, connecting to a node inside of ~~ACME's~~entity A's datacenter).

A first ssh login session LS1 is created on machine A for user X. The privileged openssh process for this login is A1 (**~~2308~~368**). Under the login session LS1, the user creates a bash shell process with PID_hash A2 (**~~2310~~369**).

At time t2 (**~~2312~~370**), inside the bash shell process A2, the user runs an ssh program under a new process A3 (**~~2314~~371**) to log in to machine B (**~~2316~~372**) with a different username ("Y"). In particular, an ssh connection is made from source IP address 2.2.2.20 and source port **10001** (Machine A's source information) to destination IP address 2.2.2.21 and destination port **22** (Machine B's destination information).

A second ssh login session LS2 is created on machine B for user Y. The privileged openssh process for this login is B1 (**~~2318~~373**). Under the login session LS2, the user creates a bash shell process with PID_hash B2 (**~~2320~~374**).

At time t3 (~~2324~~376), inside the bash shell process B2, the user runs a curl command under a new process B3 (~~2326~~377) to download a file from an external destination (~~2328~~378). In particular, an HTTPS connection is made from source IP address 2.2.2.21 and source port- **10002**- (Machine B's source information) to external destination IP address 3.3.3.30 and destination port- **443**- (the external destination's information).

Using techniques described herein, it is possible to determine the original user who initiated the connection to external destination ~~2328~~378, which in this example is a user having the username X on machine A (where the extended user session can be determined to start with ssh login session LS1).

Based on local descendant tracking, the following determinations can be on machine A and B without yet having performed additional processing (described in more detail below):

-
  - A3 is a descendant of A1 and thus associated with LS1.
  - The connection to the external domain from machine B is initiated by B3.
  - B3 is a descendant of B1 and is thus associated with LS2.
  - Connection to the external domain is thus associated with LS2.

An association between A3 and LS2 can be established based on the fact that LS2 was created based on an ssh connection initiated from A3. Accordingly, it can be determined that LS2 is a child of LS1.

To determine the user responsible for making the connection to the external destination (e.g., if it were a known bad destination), first, the process that made the connection would be traced, i.e., from B3 to LS2. Then LS2 would be traced to LS1 (i.e., LS1 is the origin login session for LS2). Thus the user for this connection is the user for LS1, i.e., X. As represented in FIG. ~~23~~ 3I, one can visualize the tracing by following the links (in the reverse direction of arrows) from external ~~source 2328~~destination 378 to A1 (~~2308~~368).

In the example scenario, it is assumed that both ssh connections occur in the same analysis period. However, the approaches described herein will also work for connections and processes that are created in different time periods.

FIG. ~~24~~ 3J illustrates an example of a process for performing extended user tracking. In various embodiments, process ~~2400~~380 is performed periodically (e.g., once an hour in a batch fashion) by- ssh tracker **148** to generate new output data. In general, batch processing allows for efficient analysis of large volumes of data. However, the approach can be adapted, as applicable, to process input data on a record-by-record fashion while maintaining the same logical data processing flow. As applicable the results of a given portion of process ~~2400~~380 are stored for use in a subsequent portion.

The process begins at ~~2402~~381 when new ssh connection records are identified. In particular, new ssh connections started during the current time period are identified by querying the connections table. The query uses filters on the start_time and dst_port columns. The values of the range filter on the start_time column are based on the current time period. The dst_port column is checked against ssh listening port(s). By default, the ssh listening port number is 22. However, as this could vary across environments, the port(s) that openssh servers are listening to in the environment can be determined by data collection agents dynamically and used as the filter value for the dst_port as applicable. In the scenario depicted in FIG. ~~23~~ 3I, the query result

will generate the records shown in FIG. ~~25A~~ 3K. Note that for the connection between machine A and B, the two machines are likely to report start_time values that are not exactly the same but close enough to be considered matching (e.g., within one minute or another appropriate amount of time). In the above table, they are shown to be the same for simplicity.

At ~~2404~~382, ssh connection records reported from source and destination sides of the same connection are matched. The ssh connection records (e.g., returned from the query at ~~2402~~381) are matched based on the following criteria:

-
  - The five tuples (src_IP, dst_IP, IP_prot, src_port, dst_port) of the connection records must match.
  - The delta between the start times of the connections must be within a limit that would account for the worst case clock difference expected between two machines in the environment and typical connection setup latency.
  - If there are multiple matches possible, then the match with the smallest time delta is chosen.

Note that record ~~2502~~390 from machine A for the incoming connection from the external source cannot be matched with another record as there is an agent only on the destination side for this connection. Example output of portion ~~2404~~382 of process ~~2400~~380 is shown in FIG. ~~25B~~ 3L. The values in the dst_PID_hash column (~~2504~~391) are that of the sshd privileged process associated with ssh logins.

At ~~2406~~383, new logins during the current time period are identified by querying the logins table. The query uses a range filter on the login_time column with values based on the current time period. In the example depicted in FIG. ~~23~~ 3I, the query result will generate the records depicted in FIG. ~~25C~~ 3M.

At ~~2408~~384, matched ssh connection records created at ~~2404~~382 and new login records created at ~~2406~~383 are joined to create new records that will eventually be stored in the login-connection table. The join condition is that dst_MID of the matched connection record is equal to the MID of the login record and the dst_PID_hash of the matched connection record is equal to the sshd_PID_hash of the login record. In the example depicted in FIG. ~~23~~ 3I, the processing performed at ~~2408~~384 will generate the records depicted in FIG. ~~25D~~ 3N.

At ~~2410~~385, login-local-descendant records in the lookback time period are identified. It is possible that a process that is created in a previous time period makes an ssh connection in the current analysis batch period. Although not depicted in the example illustrated in FIG. ~~23~~ 3I, consider a case where bash process A2 does not create ssh process A3 right away but instead that the ssh connection A3 later makes to machine B is processed in a subsequent time period than the one where A2 was processed. While processing this subsequent time period in which processes A3 and B1 are seen, knowledge of A2 would be useful in establishing that B1 is associated with A3 (via ssh connection) which is associated with A2 (via process parentage) which in turn would be useful in establishing that the parent of the second ssh login is the first ssh login. The time period for which look back is performed can be limited to reduce the amount of historical data that is considered. However, this is not a requirement (and the amount of look back can be determined, e.g., based on available processing resources). The login local descendants in the lookback time period can be identified by querying the login-local-descendant table. The query uses a range filter on the login_time column where the range is from

start_time_of_current_period-lookback_time-_to-_start_time_of_current_period. (No records as a result of performing ~~2410~~385 on the scenario depicted in FIG.~~23~~ 3I are obtained, as only a single time period is applicable in the example scenario.)

At ~~2412~~386, new processes that are started in the current time period are identified by querying the processes table. The query uses a range filter on the start_time column with values based on the current time period. In the example depicted in FIG.~~23~~ 3I, the processing performed at ~~2412~~386 will generate the records depicted in FIG.~~25E~~ 3O.

At ~~2414~~387, new login-local-descendant records are identified. The purpose is to determine whether any of the new processes in the current time period are descendants of an ssh login process and if so to create records that will be stored in the login-local-descendant table for them. In order to do so, the parent-child relationships between the processes are recursively followed. Either a top down or bottom up approach can be used. In a top down approach, the ssh local descendants in the lookback period identified at ~~2410~~385, along with new ssh login processes in the current period identified at ~~2408~~384 are considered as possible ancestors for the new processes in the current period identified at ~~2412~~386.

Conceptually, the recursive approach can be considered to include multiple sub-steps where new processes that are identified to be ssh local descendants in the current sub-step are considered as ancestors for the next step. In the example scenario depicted in FIG.~~23~~ 3I, the following descendancy relationships will be established in two sub-steps:

Sub-Step 1:

Process A2 is a local descendant of LS1 (i.e., MID=A, sshd_PID_hash=A1) because it is a child of process A1 which is the login process for LS1.

Process B2 is a local descendant of LS2 (i.e., MID=B, sshd_PID_hash=B1) because it is a child of process B1 which is the login process for LS2.

Sub-Step 2:

Process A3 is a local descendant of LS1 because it is a child of process A2 which is associated to LS1 in sub-step 1.

Process B3 is a local descendant of LS2 because it is a child of process B1 which is associated to LS2 in sub-step 1.

Implementation portion ~~2414~~387 can use a datastore that supports recursive query capabilities, or, queries can be constructed to process multiple conceptual sub-steps at once. In the example depicted in FIG.~~23~~ 3I, the processing performed at ~~2414~~387 will generate the records depicted in FIG.~~25F~~ 3P. Note that the ssh privileged processes associated with the logins are also included as they are part of the login session.

At ~~2416~~388, the lineage of new ssh logins created in the current time period is determined by associating their ssh connections to source processes that may be descendants of other ssh logins (which may have been created in the current period or previous time periods). In order to do so, first an attempt is made to join the new ssh login connections in the current period (identified at ~~2408~~384) with the combination of the login local descendants in the lookback period (identified at ~~2410~~385) and the login local descendants in the current time period (identified at ~~2412~~386). This will create adjacency relationships between child and parent logins. In the example depicted in FIG.~~23~~ 31 , the second ssh login connection will be associated with process A3 and an adjacency relationship between the two login sessions will be created (as illustrated in FIG.~~25G~~ 3Q).

Next, the adjacency relationships are used to find the original login sessions. While not shown in the sample scenario, there could be multiple ssh logins in a chain in the current time period, in

which case a recursive approach (as in ~~2414~~387) could be used. At the conclusion of portion ~~2416~~388, the login lineage records depicted in FIG. ~~25H~~ 3R will be generated. Finally, at ~~2418~~389, output data is generated. In particular, the new login-connection, login-local-descendant, and login-lineage records generated at ~~2408, 2414~~384, 387, and ~~2416~~388 are inserted into their respective output tables (e.g., in a transaction manner).

An alternate approach to matching TCP connections between machines running an agent is for the client to generate a connection GUID and send it in the connection request (e.g., the SYN packet) it sends and for the server to extract the GUID from the request. If two connection records from two machines have the same GUID, they are for the same connection. Both the client and server will store the GUID (if ~~if~~it exists) in the connection records they maintain and report. On the client side, the agent can configure the network stack (e.g~~.~~., using IP tables functionality on Linux) to intercept an outgoing TCP SYN packet and modify it to add the generated GUID as a TCP option. On the server side, the agent already extracts TCP SYN packets and thus can look for this option and extract the GUID if it exists.

~~IV. Graph-Based User Tracking~~Example graph-based user tracking and ~~Threat Detection~~ threat detection embodiments associated with data platform **12** will now be described.

Administrators and other users of network environments (e.g., ~~ACME's~~entity A's datacenter **104**) often change roles to perform tasks. As one example, suppose that at the start of a workday, an administrator (hereinafter "Joe Smith") logs in to a console, using an individualized account (e.g., username=joe.smith). Joe performs various tasks as himself (e.g., answering emails, generating status reports, writing code, etc.). For other tasks (e.g., performing updates), Joe may require different/additional permission than his individual account has (e.g., root privileges). One way Joe can gain access to such permissions is by using sudo, which will allow Joe to run a single command with root privileges. Another way Joe can gain access to such permissions is by su or otherwise logging into a shell as root. After gaining root privileges, another thing that Joe can do is switch identities. As one example, to perform administrative tasks, Joe may use "su help" or "su database-admin" to become (respectively) the help user or the database-admin user on a system. He may also connect from one machine to another, potentially changing identities along the way (e.g., logging in as joe.smith at a first console, and connecting to a database server as database-admin). When he's completed various administrative tasks, Joe can relinquish his root privileges by closing out of any additional shells created, reverting back to a shell created for user joe.smith.

While there are many legitimate reasons for Joe to change his identity throughout the day, such changes may also correspond to nefarious activity. Joe himself may be nefarious, or Joe's account (joe.smith) may have been compromised by a third party (whether an "outsider" outside of ~~ACME's~~entity A's network, or an "insider"). Using techniques described herein, the behavior of users of the environment can be tracked (including across multiple accounts and/or multiple machines) and modeled (e.g., using various graphs described herein). Such models can be used to generate alerts (e.g., to anomalous user behavior). Such models can also be used forensically, e.g., helping an investigator visualize various aspects of a network and activities that have occurred, and to attribute particular types of actions (e.g., network connections or file accesses) to specific users.

In a typical day in a datacenter, a user (e.g., Joe Smith) will log in, run various processes, and (optionally) log out. The user will typically log in from the same set of IP addresses, from IP addresses within the same geographical area (e.g., city or country), or from historically known IP addresses/geographical areas (i.e., ones the user has previously/occasionally used). A deviation

from the user's typical (or historical) behavior indicates a change in login behavior. However, it does not necessarily mean that a breach has occurred. Once logged into a datacenter, a user may take a variety of actions. As a first example, a user might execute a binary/script. Such binary/script might communicate with other nodes in the datacenter, or outside of the datacenter, and transfer data to the user (e.g., executing "curl" to obtain data from a service external to the datacenter). As a second example, the user can similarly transfer data (e.g., out of the datacenter), such as by using POST. As a third example, a user might change privilege (one or more times), at which point the user can send/receive data as per above. As a fourth example, a user might connect to a different machine within the datacenter (one or more times), at which point the user can send/receive data as per the above.

In various embodiments, the above information associated with user behavior is broken into four tiers. The tiers represent example types of information that data platform ~~102~~12 can use in modeling user behavior:

- 1. The user's entry point (e.g., domains, IP addresses, and/or geolocation information such as country/city) from which a user logs in.
- 2. The login user and machine class.
- 3. Binaries, executables, processes, etc. a user launches.
- 4. Internal servers with which the user (or any of the user's processes, child processes, etc.) communicates, and external contacts (e.g., domains, IP addresses, and/or geolocation information such as country/city) with which the user communicates (i.e., transfers data).

In the event of a security breach, being able to concretely answer questions about such information can be very important. And, collectively, such information is useful in providing an end-to-end path (e.g., for performing investigations).

In the following example, suppose a user ("UserA") logs into a machine ("Machine01") from a first IP address ("IP01"). Machine01 is inside a datacenter. UserA then launches a script ("runnable.sh") on Machine01. From Machine01, UserA next logs into a second machine ("Machine02") via ssh, also as UserA, also within the datacenter. On Machine02, UserA again launches a script ("new_runnable.sh"). On Machine02, UserA then changes privilege, becoming root on Machine02. From Machine02, UserA (now as root) logs into a third machine ("Machine03") in the datacenter via ssh, as root on Machine03. As root on Machine03, the user executes a script ("collect_data.sh") on Machine03. The script internally communicates (as root) to a MySQL-based service internal to the datacenter, and downloads data from the MySQL-based service. Finally, as root on Machine03, the user externally communicates with a server outside the datacenter ("External01"), using a POST command. To summarize what has occurred, in this example, the source/entry point is IP01. Data is transferred to an external server External01. The machine performing the transfer to External01 is Machine03. The user transferring the data is "root" (on Machine03), while the actual user (hiding behind root) is UserA.

In the above scenario, the "original user" (ultimately responsible for transmitting data to External01) is UserA, who logged in from IP01. Each of the processes ultimately started by UserA, whether started at the command line (tty) such as "runnable.sh" or started after an ssh

connection such as “new_runnable.sh,” and whether as UserA, or as a subsequent identity, are all examples of child processes which can be arranged into a process hierarchy.

As previously mentioned, machines can be clustered together logically into machine clusters. One approach to clustering is to classify machines based on information such as the types of services they provide/binaries they have installed upon them/processes they execute. Machines sharing a given machine class (as they share common binaries/services/etc.) will behave similarly to one another. Each machine in a datacenter can be assigned to a machine cluster, and each machine cluster can be assigned an identifier (also referred to herein as a machine class). One or more tags can also be assigned to a given machine class (e.g., database_servers_west or prod_web_frontend). One approach to assigning a tag to a machine class is to apply term frequency analysis (e.g., TF/IDF) to the applications run by a given machine class, selecting as tags those most unique to the class. ~~Platform 102~~Data platform 12 can use behavioral baselines taken for a class of machines to identify deviations from the baseline (e.g., by a particular machine in the class).

FIG. ~~26~~ 3S illustrates an example of a process for detecting anomalies. In various embodiments, process ~~2600~~392 is performed by data platform ~~102~~12. As explained above, a given session will have an original user. And, each action taken by the original user can be tied back to the original user, despite privilege changes and/or lateral movement throughout a datacenter. Process ~~2600~~392 begins at ~~2602~~393 when log data associated with a user session (and thus an original user) is received. At ~~2604~~394, a logical graph is generated, using at least a portion of the collected data. When an anomaly is detected (~~2606~~395), it can be recorded, and as applicable, an alert is generated (~~2608~~396). The following are examples of graphs that can be generated (e.g., at ~~2604~~394), with corresponding examples of anomalies that can be detected (e.g., at ~~2606~~395) and alerted upon (e.g., at ~~2608~~396).

~~A. Insider Behavior Graph~~

FIG. ~~27A~~ 4A illustrates a representation of an embodiment of an insider behavior graph. In the example of FIG. ~~27A~~ 4A, each node in the graph can be: (1) a cluster of users; (2) a cluster of launched processes; (3) a cluster of processes/servers running on a machine class; (4) a cluster of external IP addresses (of incoming clients); or (5) a cluster of external servers based on DNS/IP/etc. As depicted in FIG. ~~27A~~ 4A, graph data is vertically tiered into four tiers. Tier 0 (~~2752~~400) corresponds to entry point information (e.g., domains, IP addresses, and/or geolocation information) associated with a client entering the datacenter from an external entry point. Entry points are clustered together based on such information. Tier 1 (~~2754~~401) corresponds to a user on a machine class, with a given user on a given machine class represented as a node. Tier 2 (~~2756~~402) corresponds to launched processes, child processes, and/or interactive processes. Processes for a given user and having similar connectivity (e.g., sharing the processes they launch and the machines with which they communicate) are grouped into nodes. Finally, Tier 3 (~~2758~~403) corresponds to the services/servers/domains/IP addresses with which processes communicate. A relationship between the tiers can be stated as follows: Tier 0 nodes log in to tier 1 nodes. Tier 1 nodes launch tier 2 nodes. Tier 2 nodes connect to tier 3 nodes.

The inclusion of an original user in both Tier 1 and Tier 2 allows for horizontal tiering. Such horizontal tiering ensures that there is no overlap between any two users in Tier 1 and Tier 2. Such lack of overlap provides for faster searching of an end-to-end path (e.g., one starting with a Tier 0 node and terminating at a Tier 3 node). Horizontal tiering also helps in establishing baseline insider behavior. For example, by building an hourly insider behavior graph, new

edges/changes in edges between nodes in Tier 1 and Tier 2 can be identified. Any such changes correspond to a change associated with the original user. And, any such changes can be surfaced as anomalous and alerts can be generated.

As explained above, Tier 1 corresponds to a user (e.g., user "U") logging into a machine having a particular machine class (e.g., machine class "M"). Tier 2 is a cluster of processes having command line similarity (e.g., CType "C"), having an original user "U," and running as a particular effective user (e.g., user "U1"). The value of U1 may be the same as U (e.g., joe.smith in both cases), or the value of U1 may be different (e.g., U=joe.smith and U1=root). Thus, while an edge may be present from a Tier 1 node to a Tier 2 node, the effective user in the Tier 2 node may or may not match the original user (while the original user in the Tier 2 node will match the original user in the Tier 1 node).

~~As a reminder, a~~A change from a user U into a user U1 can take place in a variety of ways. Examples include where U becomes U1 on the same machine (e.g., via su), and also where U sshes to other machine(s). In both situations, U can perform multiple changes, and can combine approaches. For example, U can become U1 on a first machine, ssh to a second machine (as U1), become U2 on the second machine, and ssh to a third machine (whether as user U2 or user U3). In various embodiments, the complexity of how user U ultimately becomes U3 (or U5, etc.) is hidden from a viewer of an insider behavior graph, and only an original user (e.g., U) and the effective user of a given node (e.g., U5) are depicted. As applicable (e.g., if desired by a viewer of the insider behavior graph), additional detail about the path (e.g., an end-to-end path of edges from user U to user U5) can be surfaced (e.g., via user interactions with nodes).

FIG. ~~27B~~ 4B illustrates an example of a portion of an insider behavior graph (e.g., as rendered in a web browser). In the example shown, node ~~2702~~405 (the external IP address, 52.32.40.231) is an example of a Tier 0 node, and represents an entry point into a datacenter. As indicated by directional arrows ~~2704~~406 and ~~2706~~407, two users, "aruneli_prod" and "harish_prod," both made use of the source IP 52.32.40.231 when logging in between 5 pm and 6 pm on Sunday July 30 (~~2708~~408). Nodes ~~2710~~409 and ~~2712~~410 are examples of Tier 1 nodes, having aruneli_prod and harish_prod as associated respective original users. As previously mentioned, Tier 1 nodes correspond to a combination of a user and a machine class. In the example depicted in FIG. ~~27B~~ 4B, the machine class associated with nodes ~~2710~~409 and ~~2712~~410 is hidden from view to simplify visualization, but can be surfaced to a viewer of interface ~~2700~~404 (e.g., when the user clicks on node ~~2710~~409 or ~~2712~~410).

Nodes ~~2720-2738~~414-423 are examples of Tier 2 nodes—processes that are launched by users in Tier 1 and their child, grandchild, etc. processes. Note that also depicted in FIG. ~~27B~~ 4B is a Tier 1 node ~~2714~~411 that corresponds to a user, "root," that logged in to a machine cluster from within the datacenter (i.e., has an entry point within the datacenter). Nodes ~~2742-2744~~425-1 and 425-2 are examples of Tier 3 nodes—internal/external IP addresses, servers, etc., with which Tier 2 nodes communicate.

In the example shown in FIG. ~~27B~~ 4B, a viewer of interface ~~2700~~404 has clicked on node ~~2738~~423. As indicated in region ~~2746~~426, the user running the marathon container is "root." However, by following the directional arrows in the graph backwards from node ~~2738~~423 (i.e. from right to left), the viewer can determine that the original user, responsible for node ~~2738~~423, is "aruneli_prod," who logged into the datacenter from IP 52.32.40.231.

The following are examples of changes that can be tracked using an insider behavior graph model:

-

- A user logs in from a new IP address.
- A user logs in from a geolocation not previously used by that user.
- A user logs into a new machine class.
- A user launches a process not previously used by that user.
- A user connects to an internal server to which the user has not previously connected.
- An original user communicates with an external server (or external server known to be malicious) with which that user has not previously communicated.
- A user communicates with an external server which has a geolocation not previously used by that user.

Such changes can be surfaced as alerts, e.g., to help an administrator determine when/what anomalous behavior occurs within a datacenter. Further, the behavior graph model can be used (e.g., during forensic analysis) to answer questions helpful during an investigation. Examples of such questions include:

-
- Was there any new login activity (Tier 0) in the timeframe being investigated? As one example, has a user logged in from an IP address with unknown geolocation information? Similarly, has a user started communicating externally with a new Tier 3 node (e.g., one with unknown geolocation information).
- Has there been any suspicious login activity (Tier 0) in the timeframe being investigated? As one example, has a user logged in from an IP address that corresponds to a known bad IP address as maintained by ~~ThreatAggr~~ Threat aggregator **150**? Similarly, has there been any suspicious Tier 3 activity?
- Were any anomalous connections made within the datacenter during the timeframe being investigated? As one example, suppose a given user ("Frank") typically enters a datacenter from a particular IP address (or range of IP addresses), and then connects to a first machine type (e.g., bastion), and then to a second machine type (e.g., database_prod). If Frank has directly connected to database_prod (instead of first going through bastion) during the timeframe, this can be surfaced using the insider graph.
- Who is (the original user) responsible for running a particular process?

~~Example – Data Exfiltration~~

An example of an insider behavior graph being used in an investigation is depicted in FIGS. ~~28A~~ 4C and ~~28B~~ 4D. FIG. ~~28A~~ 4C depicts a baseline of behavior for a user, "Bill." As shown in FIG. ~~28A~~ 4C, Bill typically logs into a datacenter from the IP address, 71.198.44.40 (~~2802~~427). He typically makes use of ssh (~~2804~~428), and sudo (~~2806~~429), makes use of a set of typical applications (~~2808~~430) and connects (as root) with the external service, api.lacework.net (~~2810~~431).

Suppose Bill's credentials are compromised by a nefarious outsider ("Eve"). FIG. ~~28B~~ 4D depicts an embodiment of how the graph depicted in FIG. ~~28A~~ 4C would appear once Eve begins exfiltrating data from the datacenter. Eve logs into the datacenter (using Bill's credentials) from 52.5.66.8 (~~2852~~432). As Bill, Eve escalates her privilege to root (e.g., via su), and then becomes a different user, Alex (e.g., via su alex). As Alex, Eve executes a script, "sneak.sh" (~~2854~~433), which launches another script, "post.sh" (~~2856~~434), which contacts external server ~~2858~~435 which has an IP address of 52.5.66.7, and transmits data to it. Edges ~~2860-2866~~436-439 each represent changes in Bill's behavior. As previously mentioned, such changes can be detected as anomalies and associated alerts can be generated. As a first example, Bill logging in from an IP address he has not previously logged in from (~~2860~~436) can generate an alert. As a second example, while Bill does typically make use of sudo (~~2806~~429), he has not previously executed sneak.sh (~~2854~~433) or post.sh (~~2856~~434) and the execution of those scripts can generate alerts as well. As a third example, Bill has not previously communicated with server ~~2858~~435, and an alert can be generated when he does so (~~2866~~439). Considered individually, each of edges ~~2860-2866~~436-439 may indicate nefarious behavior, or may be benign. As an example of a benign edge, suppose Bill begins working from a home office two days a week. The first time he logs in from his home office (i.e., from an IP address that is not 71.198.44.40), an alert can be generated that he has logged in from a new location. Over time, however, as Bill continues to log in from his home office but otherwise engages in typical activities, Bill's graph will evolve to include logins from both 71.198.44.40 and his home office as baseline behavior. Similarly, if Bill begins using a new tool in his job, an alert can be generated the first time he executes the tool, but over time will become part of his baseline.

In some cases, a single edge can indicate a serious threat. For example, if server ~~2852~~432 (or ~~2858~~435) is included in a known bad IP listing, edge ~~2860~~436 (or ~~2866~~439) indicates compromise. An alert that includes an appropriate severity level (e.g., "threat level high") can be generated. In other cases, a combination of edges could indicate a threat (where a single edge might otherwise result in a lesser warning). In the example shown in FIG. ~~28B~~ 4D, the presence of multiple new edges is indicative of a serious threat. Of note, even though "sneak.sh" and "post.sh" were executed by Alex, because data platform ~~10~~212 also keeps track of an original user, the compromise of ~~Bob's~~user B's account will be discovered.

~~B. User Login Graph~~

FIG. ~~29~~ 4E illustrates a representation of an embodiment of a user login graph. In the example of FIG. ~~29~~ 4E, tier 0 (~~2902~~440) clusters source IP addresses as belonging to a particular country (including an "unknown" country) or as a known bad IP. Tier 1 (~~2904~~441) clusters user logins, and tier 2 (~~2906~~442) clusters type of machine class into which a user is logging in. The user login graph tracks the typical login behavior of users. By interacting with a representation of the graph, answers to questions such as the following can be obtained:

-
  - Where is a user logging in from?
  - Have any users logged in from a known bad address?
  - Have any non-developer users accessed development machines?
  - Which machines does a particular user access?

Examples of alerts that can be generated using the user login graph include:

-
  - A user logs in from a known bad IP address.
  - A user logs in from a new country for the first time.
  - A new user logs into the datacenter for the first time.
  - A user accesses a machine class that the user has not previously accessed.

~~C. Privilege Change Graph~~

One way to track privilege changes in a datacenter is by monitoring a process hierarchy of processes. To help filter out noisy commands/processes such as "su -u," the hierarchy of processes can be constrained to those associated with network activity. In a *nix system, each process has two identifiers assigned to it, a process identifier (PID) and a parent process identifier (PPID). When such a system starts, the initial process is assigned a PID 0. Each user process has a corresponding parent process.

~~FIG. 30 illustrates a representation of a process tree. In the example shown in FIG. 30, PIDs have been replaced with an effective user running a given process. Thus, a designation of "root" (3002) indicates the user running the process is root, and a designation of "avahi" (3004) indicates the user running the process is avahi. Further, in the example shown in FIG. 30, processes depicted to the right of other processes are child processes. In line 3006, the user avahi became root and ran the process "padae_run," whose parent is "avahi-daemon." This represents a privilege change (from avahi to root).~~

Using techniques described herein, a graph can be constructed (also referred to herein as a privilege change graph) which models privilege changes. In particular, a graph can be constructed which identifies where a process P1 launches a process P2, where P1 and P2 each have an associated user U1 and U2, with U1 being an original user, and U2 being an effective user. In the graph, each node is a cluster of processes (sharing a CType) executed by a particular (original) user. As all the processes in the cluster belong to the same user, a label that can be used for the cluster is the user's username. An edge in the graph, from a first node to a second node, indicates that a user of the first node changed its privilege to the user of the second node. FIG. ~~31~~ 4F illustrates an example of a privilege change graph. In the example shown in FIG. ~~31~~ 4F, each node (e.g., nodes ~~3102~~444 and ~~3104~~445) represents a user. Privilege changes are indicated by edges, such as edge ~~3106~~446.

As with other graphs, anomalies in graph ~~3100~~443 can be used to generate alerts. Three examples of such alerts are as follows:

-
  - New user entering the datacenter. Any time a new user enters the datacenter and runs a process, the graph will show a new node, with a new CType. This indicates a new user has been detected within the datacenter. FIG. ~~31~~ 4F is a representation of an example of an interface that depicts such an alert. Specifically, as indicated in region ~~3108~~447, an alert for the time period 1 pm-2 pm on June 8 was generated. The alert identifies that a new user, Bill (~~3110~~448) executed a process.
  - Privilege change. As explained above, a new edge, from a first node (user A) to a second node (user B) indicates that user A has changed privilege to user B.

- Privilege escalation. Privilege escalation is a particular case of privilege change, in which the first user becomes root.

An example of an anomalous privilege change and an example of an anomalous privilege escalation are each depicted in graph ~~3200~~450 of FIG.~~ 32~~ 4G. In particular, as indicated in region ~~3202~~451, two alerts for the time period 2 pm-3 pm on June 8 were generated (corresponding to the detection of the two anomalous events). In region ~~3204~~452, root has changed privilege to the user "daemon," which root has not previously done. This anomaly is indicated to the user by highlighting the daemon node (e.g., outlining it in ~~the~~a particular color, e.g., red). As indicated by edge ~~3206~~453, Bill has escalated his privilege to the user root (which can similarly be highlighted in region ~~3208~~454). This action by Bill represents a privilege escalation.

~~V.~~An Extensible ~~Query Interface~~query interface for ~~Dynamic Data Compositions~~dynamic data compositions and ~~Filter Applications~~filter applications will now be described.

As described ~~throughout this Specification~~herein, datacenters are highly dynamic environments. And, different customers of data platform ~~10~~12 (e.g., ~~ACME~~entity A vs. ~~BETA~~entity B) may have different/disparate needs/requirements of data platform ~~10~~12, e.g., due to having different types of assets, different applications, etc. Further, as time progresses, new software tools will be developed, new types of anomalous behavior will be possible (and should be detectable), etc. In various embodiments, data platform ~~10~~12 makes use of predefined relational schema (including by having different predefined relational schema for different customers). However, the complexity and cost of maintaining/updating such predefined relational schema can rapidly become problematic—particularly where the schema includes a mix of relational, nested, and hierarchical (graph) datasets. In other embodiments, the data models and filtering applications used by data platform ~~10~~12 are extensible. As will be described in more detail below, in various embodiments, data platform ~~10~~12 supports dynamic query generation by automatic discovery of join relations via static or dynamic filtering key specifications among composable data sets. This allows a user of data platform ~~10~~12 to be agnostic to modifications made to existing data sets as well as creation of new data sets. The extensible query interface also provides a declarative and configurable specification for optimizing internal data generation and derivations.

As will also be described in more detail below, data platform ~~10~~12 is configured to dynamically translate user interactions (e.g., received via web app **120**) into SQL queries (and without the user needing to know how to write queries). Such queries can then be performed (e.g., by query service **166**) against any compatible backend (e.g., ~~database 142~~data store **30**).

~~A. Visualization Examples~~

FIG.~~ 33~~ 4H illustrates an example of a user interacting with a portion of an interface. When a user visits data platform ~~10~~12 (e.g., via web app **120** using a browser), data is extracted from ~~database 142~~data store **30** as needed (e.g., by query service **166**), to provide the user with information, such as the visualizations depicted variously ~~throughout the Specification (e.g., in FIGS. 9 and 33).~~herein). As the user continues to interact with such visualizations (e.g., clicking on graph nodes, entering text into search boxes, navigating between tabs (e.g.,- tab ~~3302~~455 vs. ~~3322~~465)), such interactions act as triggers that cause query service **166** to continue to obtain information from ~~database 142~~data store **30** as needed (and as described in more detail below).

In the example shown in FIG.~~ 33, ACME administrator Alice~~ 4H, user A is viewing a dashboard that provides various information about ~~ACME~~entity A users (~~3302~~455), during the time period March 2 at midnight-March 25 at 7 pm (which she selected by interacting with region ~~3304~~456).

Various statistical information is presented to ~~Alice~~user A in region ~~3306~~457. Region ~~3308~~458 presents a timeline of events that occurred during the selected time period. ~~Alice~~User A has opted to list only the critical, high, and medium events during the time period by clicking on the associated boxes (~~3310-3314~~459-461). A total of 55 low severity, and- 155- info-only events also occurred during the time period. Each time ~~Alice~~user A interacts with an element in FIG. ~~33~~ 4H (e.g., clicks on box ~~3314~~461, clicks on- link ~~3320~~464-1, or clicks on tab ~~3322~~465), her actions are translated/formalized into filters on the data set and used to dynamically generate SQL queries. The SQL queries are generated transparently to ~~Alice~~user A (and also to a designer of the user interface shown in FIG. ~~33~~ 4H). ~~Alice~~User A notes in the timeline (~~3316~~462) that a user, Harish, connected to a known bad server (examplebad.com) using wget, an event that has a critical severity level. ~~Alice~~User A can click on region ~~3318~~463 to expand details about the event inline (which will display, for example, the text "External connection made to known bad host examplebad.com at port **80**- from application 'wget' running on host ~~devl~~dev1.lacework.internal as user harish") directly below line ~~3316. Alice~~462. User A can also click on ~~region 3320~~ link 464-1, which will take her to a dossier for the event (depicted in FIG. ~~34~~ 4I). As will be described in more detail below, a dossier is a template for a collection of visualizations.

As shown in interface ~~3400~~466, the event of Harish using wget to contact examplebad.com on March 16 was assigned an event ID of **9291** by data platform ~~102 (3402~~12 (467). For convenience to ~~Alice~~user A, the event is also added to her dashboard in region ~~3420~~476 as a bookmark (~~3404~~468). A summary of the event is depicted in region ~~3406~~469. By interacting with boxes shown in region ~~3408, Alice~~470, user A can see a timeline of related events. In this case, ~~Alice~~user A has indicated that she would like to see other events involving the wget application (by clicking box ~~3410~~471). Events of critical and medium security involving wget occurred during the one hour window selected in region ~~3412~~472.

Region ~~3414~~473 automatically provides ~~Alice~~user A with answers to questions that may be helpful to have answers to while investigating event **9291**. If ~~Alice~~user A clicks on any of the links in the event description (~~3416~~474), she will be taken to a corresponding dossier for the link. As one example, suppose ~~Alice~~user A clicks on link ~~3418~~475. She will then be presented with interface ~~3500~~477 shown in FIG. ~~35~~ 4J.

Interface ~~3500~~477 is an embodiment of a dossier for a domain. In this example, the domain is "examplebad.com," as shown in region ~~3502~~478. Suppose ~~Alice~~user A would like to track down more information about interactions ~~ACME~~entity A resources have made with examplebad.com between January 1 and March 20. She selects the appropriate time period in region ~~3504~~479 and information in the other portions of interface ~~3500~~477 automatically update to provide various information corresponding to the selected time frame. As one example, ~~Alice~~user A can see that contact was made with examplebad.com a total of 17 times during the time period (~~3506~~480), as well as a list of each contact (~~3508~~481). Various statistical information is also included in the dossier for the time period (~~3510~~482). If she scrolls down in interface ~~3500, Alice~~477, user A will be able to view various polygraphs associated with examplebad.com, such as an application-communication polygraph (~~3512~~483).

~~B. Query Service Data Model~~

Data stored in ~~database 142~~data store **30** can be internally organized as an activity graph. In the activity graph, nodes are also referred to as Entities. Activities generated by Entities are modeled as directional edges between nodes. Thus, each edge is an activity between two Entities. One example of an Activity is a "login" Activity, in which a user Entity logs into a machine Entity

(with a directed edge from the user to the machine). A second example of an Activity is a "launch" Activity, in which a parent process launches a child process (with a directed edge from the parent to the child). A third example of an Activity is a "DNS query" Activity, in which either a process or a machine performs a query (with a directed edge from the requestor to the answer, e.g., an edge from a process to www.example.com). A fourth example of an Activity is a network "connected to" Activity, in which processes, IP addresses, and listen ports can connect to each other (with a directed edge from the initiator to the server).

As will be described in more detail below, query service **166** provides either relational views or graph views on top of data stored in ~~database **142.**~~data store **30**. Typically, a user will ~~to see~~ want to see data filtered using the activity graph. For example, if an entity was not involved in an activity in a given time period, that entity should be filtered out of query results. Thus, a request to show "all machines" in a given time frame will be interpreted as "show distinct machines that were active" during the time frame.

Query service **166** relies on three main data model elements: fields, entities, and filters. As used herein, a field is a collection of values with the same type (logical and physical). A field can be represented in a variety of ways, including: 1. a column of relations (table/view), 2. a return field from another entity, 3. an SQL aggregation (e.g., COUNT, SUM, etc.), 4. an SQL expression with the references of other fields specified, and 5. a nested field of a JSON object. As viewed by query service **166**, an entity is a collection of fields that describe a data set. The data set can be composed in a variety of ways, including: 1. a relational table, 2. a parameterized SQL statement, 3. DynamicSQL created by a Java function, and 4. join/project/aggregate/subclass of other entities. Some fields are common for all entities. One example of such a field is a "first observed" timestamp (when first use of the entity was detected). A second example of such a field is the entity classification type (e.g., one of: 1. Machine (on which an agent is installed), 2. Process, 3. Binary, 4. UID, 5. IP, 6. DNS Information, 7. ListenPort, and 8. PType). A third example of such a field is a "last observed" timestamp.

A filter is an operator that: 1. takes an entity and field values as inputs, 2. a valid SQL expression with specific reference(s) of entity fields, or 3. is a conjunct/disjunct of filters. As will be described in more detail below, filters can be used to filter data in various ways, and limit data returned by query service **166** without changing the associated data set.

~~1. Cards~~

As mentioned above, a dossier is a template for a collection of visualizations. Each visualization (e.g., the box including chart ~~35~~**14484**) has a corresponding card, which identifies particular target information needed (e.g., from ~~database **142**~~data store **30**) to generate the visualization. In various embodiments, data platform ~~**102**~~**12** maintains a global set of dossiers/cards. Users of data platform ~~**102**~~**12** such as ~~Alice~~user A can build their own dashboard interfaces using preexisting dossiers/cards as components, and/or they can make use of a default dashboard (which incorporates various of such dossiers/cards).

~~a. Card Specification~~

A JSON file can be used to store multiple cards (e.g., as part of a query service catalog). A particular card is represented by a single JSON object with a unique name as a field name. ~~For example:~~

~~{~~

~~• —~~

~~○ "Card1": {~~

- ...
- },
- "Card2": {
  - ...
- }

}

Each card ~~is~~may be described by the following named fields:
TYPE: the type of the card. Example values include:

- Entity (the default type)
- SQL
- Filters
- DynamicSQL

~~GraphFilter~~
~~Graph~~

- graphFilter
- graph
- Function
- Template

PARAMETERS: a JSON array object that contains an array of parameter objects with the following fields:

- name (the name of the parameter)
- required (a Boolean flag indicating whether the parameter is required or not)
- default (a default value of the parameter)
- props (a generic JSON object for properties of the parameter. Possible values are: "utype" (a user defined type), and "scope" (an optional property to configure a namespace of the parameter))
- value (a value for the parameter—non-null to override the default value defined in nested source entities)

SOURCES: a JSON array object explicitly specifying references of input entities. Each source reference has the following attributes:

-
  - name (the card/entity name or fully-qualified Table name)
  - type (required for base Table entity)
  - alias (an alias to access this source entity in other fields (e.g., returns, filters, groups, etc))

RETURNS: a required JSON array object of a return field object. A return field object can be described by the following attributes:

-
  - field (a valid field name from a source entity)
  - expr (a valid SQL scalar expression. References to input fields of source entities are specified in the format of #{Entity. Field}. Parameters can also be used in the expression in the format of ~~$ {~~${ParameterName})
  - type (the type of field, which is required for return fields specified by expr. It is also required for all return fields of an Entity with an SQL type)
  - alias (the unique alias for return field)
  - aggr (possible aggregations are: COUNT, COUNT_DISTINCT, DISTINCT, MAX, MIN, AVG, SUM, FIRST_VALUE, LAST_VALUE)
  - case (JSON array object represents conditional expressions “when” and “expr”)
  - fieldsFrom, and, except (specification for projections from a source entity with excluded fields)
  - props (general JSON object for properties of the return field. Possible properties include: “filterGroup,” “title,” “format,” and “utype”)

PROPS: generic JSON objects for other entity properties
SQL: a JSON array of string literals for SQL statements. Each string literal can contain parameterized expressions ~~$~~{$ {ParameterName} and/or composable entity by # ~~{EntityName}~~#{Entity Name} GRAPH: required for graph entity. Has the following required fields:

- ~~GRAPH: required for Graph entity. Has the following required fields:~~
  - source (including “type,” “props,” and “keys”)
  - target (including “type,” “props,” and “keys”)
  - edge (including “type” and “props”)

JOINS: a JSON array of join operators. Possible fields for a join operator include:

-

- type (possible join types include: “loj”—Left Outer Join, “join”—Inner Join, “in”—Semi Join, “implicit”—Implicit Join)
- left (a left hand side field of join)
- right (a right hand side field of join)
- keys (key columns for multi-way joins)
- order (a join order of multi-way joins)

FKEYS: a JSON array of FilterKey(s). The fields for a FilterKey are:

-
  - type (type of FilterKey)
  - fieldRefs (reference(s) to return fields of an entity defined in the sources field)
  - alias (an alias of the FilterKey, used in implicit join specification)

FILTERS: a JSON array of filters (conjunct). Possible fields for a filter include:

-
  - type (types of filters, including: “eq”—equivalent to SQL=, “ne”—equivalent to SQL < >, “ge”—equivalent to SQL >=, “gt”—equivalent to SQL >, “le”—equivalent to SQL <=, “lt”—equivalent to SQL <, “like”—equivalent to SQL LIKE, “not_like”—equivalent to SQL NOT LIKE, “rlike”—equivalent to SQL RLIKE (Snowflake specific), “not_rlike”—equivalent to SQL NOT RLIKE (Snowflake specific), “in”—equivalent to SQL IN, “not_in”—equivalent to SQL NOT IN)
  - expr (generic SQL expression)
  - field (field name)
  - value (single value)
  - values (for both IN and NOT IN)

ORDERS: a JSON array of ORDER BY for returning fields. Possible attributes for the ORDER BY clause include:

-
  - field (field ordinal index (1 based) or field alias)
  - order (asc/desc, default is ascending order)

GROUPS: a JSON array of GROUP BY for returning fields. Field attributes are:

-
  - field (ordinal index (1 based) or alias from the return fields)

LIMIT: a limit for the number of records to be returned
OFFSET: an offset of starting position of returned data. Used in combination with limit for pagination.

~~b. Extensibility Examples~~

Suppose customers of data platform ~~102~~12 (e.g., ~~ACME~~entity A and ~~BETA~~entity B) request new data transformations or a new aggregation of data from an existing data set (as well as a corresponding visualization for the newly defined data set). As mentioned above, the data models and filtering applications used by data platform ~~102~~12 are extensible. Thus, two example scenarios of extensibility are (1) extending the filter data set, and (2) extending a FilterKey in the filter data set.

~~Platform 102~~Data platform 12 includes a query service catalog that enumerates cards available to users of data platform ~~102~~12. New cards can be included for use in data platform ~~102~~12 by being added to the query service catalog (e.g., by an operator of data platform ~~102~~12). For reusability and maintainability, a single external-facing card (e.g., available for use in a dossier) can be composed of multiple (nested) internal cards. Each newly added card (whether external or internal) will also have associated FilterKey(s) defined. A user interface (UI) developer can then develop a visualization for the new data set in one or more dossier templates. The same external card can be used in multiple dossier templates, and a given external card can be used multiple times in the same dossier (e.g., after customization). Examples of external card customization include customization via parameters, ordering, and/or various mappings of external data fields (columns).

~~FIG. 36 illustrates an example of a card specification. The data set associated with the card shows a list of binary information running by users within the period of [StartTimeRange (3602), EndTimeRange (3604)]. The card depicted in FIG. 36 is implemented by extending an existing (base) card, MVIEW_INTERNAL.ENTITY_VIEW_T (as shown in sources field 3606) with explicit filters on ENTITY_TYPE (as shown in filters field 3608). It also adds dynamic filters via fkeys PID_KEY (3610) and implicit join.~~

~~The ProcessClusterFilters Entity is a global external filter data set comprising composable internal data sets (which use the same syntax as cards such as the card depicted in FIG. 36). The ProcessClusterFilters Entity defines a logical group of entities, the primary purpose of which is to form the scope of filters. Physical grouping will be determined at runtime by filters provided by users. The filters and grouping are referred to herein as implicit filters and joins, respectively. Similar to other derived entities, ProcessClusterFilters has the following standard JSON fields, with extended definitions:~~

~~sources: an array of references to entities. A given reference can be either a base or a derived entity.~~

~~returns: an array of external filter names that represent the schema of ProcessClusterFilters.~~

~~fkeys: an array of FilterKey(s).~~

~~joins: an array of implicit joins. Each implicit join defines a group of FilterKey(s) with the same key type. The same FilterKey may be used in different join groups.~~

~~Returning to FIG. 36, the dynamic application of filters via implicit join illustrates that the definition of the card (referred to as "Card189" in the query service catalog) is transparent to future changes in ProcessClusterFilters.~~

As mentioned above, a second extensibility scenario is one in which a FilterKey in the filter data set is extended (i.e., existing template functions are used to define a new data set). As also mentioned above, data sets used by data platform ~~102~~12 are composable/reusable/extensible,

irrespective of whether the data sets are relational or graph data sets. One example data set is the User Tracking polygraph, which is generated as a graph data set (comprising nodes and edges). Like other polygraphs, User Tracking is an external data set that can be visualized both as a graph (via the nodes and edges) and can also be used as a filter data set for other cards, via the cluster identifier (CID) field.

~~An example of this scenario is depicted in FIG. 37, in which three new user tracking data sets are added as new data sources for the ProcessClusterFilters Entity. Corresponding CID fields (UserTrackingIPAddress_CID, UserTrackingUser_CID, and UserTrackingDNSSep_CID) are also added as external filters to ProcessClusterFilters. Note that while changes will be made to ProcessClusterFilters, due to the extensibility techniques used herein, such changes will not impact card **3600**.~~

~~2. Generating SQL Queries~~

As mentioned above, as users such as ~~Alice~~user A navigate through/interact with interfaces provided by data platform **~~10~~212** (e.g., as shown in FIG.~~ 33~~ 4H), such interactions trigger query service **166** to generate and perform queries against ~~database **142**~~data store **30**. Dynamic composition of filter datasets can be implemented using FilterKeys and FilterKey Types. A FilterKey can be defined as a list of columns and/or fields in a nested structure (e.g., JSON). Instances of the same FilterKey Type can be formed as an Implicit Join Group. The same instance of a FilterKey can participate in different Implicit Join Groups. A list of relationships among all possible Implicit Join Groups is represented as a Join ~~Graph~~graph for the entire search space to create a final data filter set by traversing edges and producing Join Path(s).

~~a. Introspection~~

Each card (e.g., as stored in the query service catalog and used in a dossier) can be introspected by a/card/describe/CardID REST request.~~ FIGS. 38 and 39 both depict examples of card schema introspection. In particular, FIG. 38 depicts introspection of a non-graph schema, while FIG. 39 depicts introspection of a graph schema. In both cases, a request is made (via the REST API) for information pertinent to a particular card, in accordance with the specified schema, and subject to the specified filters.~~

~~b. Join~~

At runtime (e.g., whenever it receives a request from web ~~frontend~~app **120**), query service **166** parses the list of implicit joins and creates a Join ~~Graph~~graph to manifest relationships of FilterKeys among Entities. A Join ~~Graph~~graph (an example of which is depicted in FIG.~~ 40~~ 4K) comprises a list of Join Link(s). A Join Link represents each implicit join group by the same FilterKey type. A Join Link maintains a reverse map (Entity-to-FilterKey) of FilterKeys and their Entities. As previously mentioned, Entities can have more than one FilterKey defined. The reverse map guarantees one FilterKey per Entity can be used for each JoinLink. Each JoinLink also maintains a list of entities for the priority order of joins. Each JoinLink is also responsible for creating and adding directional edge(s) to graphs. An edge represents a possible join between two Entities.

At runtime, each Implicit Join uses the Join ~~Graph~~graph to find all possible join paths. The search of possible join paths starts with the outer FilterKey of an implicit join. One approach is to use a shortest path approach, with breadth first traversal and subject to the following criteria:

-
  - Use the priority order list of Join Links for all entities in the same implicit join group.

- Stop when a node (Entity) is reached which has local filter(s).
- Include all join paths at the same level (depth).
- Exclude join paths based on the predefined rules (path of edges).

~~e. Examples~~

FIG.~~ 41 depicts an example introspection corresponding to the card depicted in FIG. 36. In particular, in FIG. 41, the card is filtered by an external DNS name. FIG. 42 depicts an example query service log for a Join Path search (in accordance with the breadth first approach described above) corresponding to FIG. 41. The shortest path is indicated at line **4202**. Finally, the SQL generated by query service **166**, using the result of the Join Path search, is depicted in FIG. 43. FIG. 44 depicts an example introspection corresponding to FIG. 37. In particular, in FIG. 44, filtering on Card **189** is performed by the User Tracking cluster. FIG. 45 depicts an example query service log for a Join Path search corresponding to FIG. 44. Finally, the SQL generated by query service **166**, using the result of the Join Path search, is depicted in FIG. 46. Examples of translating a filter and a join, respectively, to an SQL predicate are depicted in FIGS. 47A and 47B, and in FIGS. 47C and 47D.~~

~~FIG. 48~~ 4L illustrates an example of a process for dynamically generating and executing a query. In various embodiments,~~ process **4800**485 is performed by data platform **102**12. The process begins at **4802**486 when a request is received to filter information associated with activities within a network environment. One example of such a request occurs in response to ~~Alice~~user A clicking on tab **3322**465. Another example of such a request occurs in response to ~~Alice~~user A clicking on~~ icon **3320**~~ link **464-1**. Yet another example of such a request occurs in response to ~~Alice~~user A clicking on~~ icon **3324**~~ link **464-2** and selecting (e.g., from a dropdown) an option to filter (e.g., include, exclude) based on specific criteria that she provides (e.g., an IP address, a username, a range of criteria, etc.).

At **4804**487, a query is generated based on an implicit join. One example of processing that can be performed at **4804**487 is as follows. As explained above, one way dynamic composition of filter datasets can be implemented is by using FilterKeys and FilterKey Types. And, instances of the same FilterKey Type can be formed as an Implicit Join Group. A Join ~~Graph~~graph for the entire search space can be constructed from a list of all relationships among all possible Join Groups. And, a final data filter set can be created by traversing edges and producing one or more Join Paths. Finally, the shortest path in the join paths is used to generate an SQL query string. One approach to generating an SQL query string is to use a query building library (authored in an appropriate language such as Java).~~ FIG. 49A illustrates a code excerpt of~~ For example, a common interface "sqlGen" ~~(included in various embodiments in the query building library used by query service **166**). An example way that sqlGen can~~may be used in conjunction with~~ process **4800**485 is as follows. First, a card/entity is composed by a list of input cards/entities, where each input card recursively is composed by its own list of input cards. This nested structure can be visualized as a tree of query blocks (SELECT) in standard SQL constructs. SQL generation can be performed as the traversal of the tree from root to leaf entities (top-down), calling the sqlGen of each entity. Each entity can be treated as a subclass of the Java class (Entity). An implicit join filter (EntityFilter) is implemented as a subclass of Entity, similar to the right hand side of a SQL semi-join operator. Unlike the static SQL semi-join construct, it is conditionally and recursively generated even if it is specified in the input sources of the JSON specification. Another recursive interface can also be used in conjunction with~~ process **4800**485, preSQLGen, which is primarily the entry point for EntityFilter to run a search and generate

nested implicit join filters. During preSQLGen recursive invocations, the applicability of implicit join filters is examined and pushed down to its input subquery list. Another top-down traversal, pullUpCachable, can be used to pull up common sub-query blocks, including those dynamically generated by preSQLGen, such that SELECT statements of those cacheable blocks are generated only once at top-level WITH clauses. A recursive interface, sqlWith, is used to generate nested subqueries inside WITH clauses. The recursive calls of a sqlWith function can generate nested WITH clauses as well. An sqlFrom function can be used to generate SQL FROM clauses by referencing those subquery blocks in the WITH clauses. It also produces INNER/OUTER join operators based on the joins in the specification. Another recursive interface, sqlWhere, can be used to generate conjuncts and disjuncts of local predicates and semi-join predicates based on implicit join transformations. Further, sqlProject, sqlGroupBy, sqlOrderBy, and sqlLimitOffset can respectively be used to translate the corresponding directives in JSON spec to SQL SELECT list, GROUP BY, ORDER BY, and LIMIT/OFFSET clauses.

Returning to process ~~4800~~485, at ~~4806~~488, the query (generated at ~~4804~~487) is used to respond to the request. As one example of the processing performed at ~~4806~~488, the generated query is used to query ~~database 142~~data store **30** and provide (e.g., to web app **120**) fact data formatted in accordance with a schema (e.g., as associated with a card associated with the request received at ~~4802~~486).

~~d. Additional Examples~~

~~FIG. 49B illustrates three examples of SQL entities. An SQL entity is composable using other entities in its statement, as well as input to other entities.~~

~~FIGS. 50A and 50B illustrate, collectively, example portions of an embodiment of a ProcessClusterFilters definition.~~

~~FIG. 51 illustrates an example of an introspection corresponding to a ProcessClusterFilters request. As mentioned above, introspection can be performed by a /card/describe/CardID REST request—a dynamic card parsed and created by query service **166**. The sources of the request are a target card (e.g., Card26) and other filter activity cards (e.g., ProcessActivity) referenced by the fields (ProcessClusterFilters.USERNAME and ProcessClusterFilters.EXE_PATH) in the filters.~~

~~FIG. 52A illustrates an example of a base table card. FIG. 52B illustrates an example of a filter request, and FIG. 52C illustrates an example of an SQL query (e.g., generated by query service **166** in response to the request). FIG. 52D illustrates an additional example of a filter request, and FIG. 52E illustrates a corresponding dynamically generated SQL query.~~

~~FIGS. 53A, 53B, and 53C illustrate, respectively, a card, a request, and a generated SQL query corresponding to a join. FIGS. 53D and 53E illustrate, respectively, a request and a generated SQL query corresponding to an explicit filter join.~~

~~Although the foregoing embodiments have been described in some detail for purposes of clarity of understanding, the invention is not limited to the details provided. There are many alternative ways of implementing the invention. The disclosed embodiments are illustrative and not restrictive.~~

Although the examples described herein largely relate to embodiments where data is collected from agents and ultimately stored in a data store such as those provided by Snowflake, in other embodiments data that is collected from agents and other sources may be stored in different ways. For example, data that is collected from agents and other sources may be stored in a data warehouse, data lake, data mart, and/or any other data store.

A data warehouse may be embodied as an analytic database (e.g., a relational database) that is created from two or more data sources. Such a data warehouse may be leveraged to store

historical data, often on the scale of petabytes. Data warehouses may have compute and memory resources for running complicated queries and generating reports. Data warehouses may be the data sources for business intelligence ('B1') systems, machine learning applications, and/or other applications. By leveraging a data warehouse, data that has been copied into the data warehouse may be indexed for good analytic query performance, without affecting the write performance of a database (e.g., an Online Transaction Processing ('OLTP') database). Data warehouses also enable the joining data from multiple sources for analysis. For example, a sales OLTP application probably has no need to know about the weather at various sales locations, but sales predictions could take advantage of that data. By adding historical weather data to a data warehouse, it would be possible to factor it into models of historical sales data.

Data lakes, which store files of data in their native format, may be considered as "schema on read" resources. As such, any application that reads data from the lake may impose its own types and relationships on the data. Data warehouses, on the other hand, are "schema on write," meaning that data types, indexes, and relationships are imposed on the data as it is stored in the EDW. "Schema on read" resources may be beneficial for data that may be used in several contexts and poses little risk of losing data. "Schema on write" resources may be beneficial for data that has a specific purpose, and good for data that must relate properly to data from other sources. Such data stores may include data that is encrypted using homomorphic encryption, data encrypted using privacy-preserving encryption, smart contracts, non-fungible tokens, decentralized finance, and other techniques.

Data marts may contain data oriented towards a specific business line whereas data warehouses contain enterprise-wide data. Data marts may be dependent on a data warehouse, independent of the data warehouse (e.g., drawn from an operational database or external source), or a hybrid of the two. In embodiments described herein, different types of data stores (including combinations thereof) may be leveraged. Such data stores may be proprietary or may be embodied as vendor provided products or services such as, for example, Google BigQuery, Druid, Amazon Redshift, IBM Db2, Dremio, Databricks Lakehouse Platform, Cloudera, Azure Synapse Analytics, and others.

The deployments (e.g., a customer's cloud deployment) that are analyzed, monitored, evaluated, or otherwise observed by the systems described herein (e.g., systems that include components such as the platform **12** of FIG. **1**D, the data collection agents described herein, and/or other components) may be provisioned, deployed, and/or managed using infrastructure as code ('IaC'). IaC involves the managing and/or provisioning of infrastructure through code instead of through manual processes. With IaC, configuration files may be created that include infrastructure specifications. IaC can be beneficial as configurations may be edited and distributed, while also ensuring that environments are provisioned in a consistent manner. IaC approaches may be enabled in a variety of ways including, for example, using IaC software tools such as Terraform by HashiCorp. Through the usage of such tools, users may define and provide data center infrastructure using JavaScript Object Notation ('JSON'), YAML, proprietary formats, or some other format. In some embodiments, the configuration files may be used to emulate a cloud deployment for the purposes of analyzing the emulated cloud deployment using the systems described herein. Likewise, the configuration files themselves may be used as inputs to the systems described herein, such that the configuration files may be inspected to identify vulnerabilities, misconfigurations, violations of regulatory requirements, or other issues. In fact, configuration files for multiple cloud deployments may even be used by the systems described herein to identify best practices, to identify configuration files that deviate from typical

configuration files, to identify configuration files with similarities to deployments that have been determined to be deficient in some way, or the configuration files may be leveraged in some other ways to detect vulnerabilities, misconfigurations, violations of regulatory requirements, or other issues prior to deploying an infrastructure that is described in the configuration files. In some embodiments the techniques described herein may be use in multi-cloud, multi-tenant, cross-cloud, cross-tenant, cross-user, industry cloud, digital platform, and other scenarios depending on specific need or situation.

In some embodiments, the deployments that are analyzed, monitored, evaluated, or otherwise observed by the systems described herein (e.g., systems that include components such as the platform **12** of FIG. **1**D, the data collection agents described herein, and/or other components) may be monitored to determine the extent to which a particular component has experienced "drift" relative to its associated IaC configuration. Discrepancies between how cloud resources were defined in an IaC configuration file and how they are currently configured in runtime may be identified and remediation workflows may be initiated to generate an alert, reconfigure the deployment, or take some other action. Such discrepancies may occur for a variety of reasons. Such discrepancies may occur, for example, due to maintenance operations being performed, due to incident response tasks being carried out, or for some other reason. Readers will appreciate that while IaC helps avoid initial misconfigurations of a deployment by codifying and enforcing resource creation, resource configuration, security policies, and so on, the systems described herein may prevent unwanted drift from occurring during runtime and after a deployment has been created in accordance with an IaC configuration.

In some embodiments, the deployments (e.g., a customer's cloud deployment) that are analyzed, monitored, evaluated, or otherwise observed by the systems described herein (e.g., systems that include components such as the platform **12** of FIG. **1**D, the data collection agents described herein, and/or other components) may also be provisioned, deployed, and/or managed using security as code ('SaC'). SaC extends IaC concepts by defining cybersecurity policies and/or standards programmatically, so that the policies and/or standards can be referenced automatically in the configuration scripts used to provision cloud deployments. Stated differently, SaC can automate policy implementation and cloud deployments may even be compared with the policies to prevent "drift." For example, if a policy is created where all personally identifiable information ('PII') or personal health information ('PHI') must be encrypted when it is stored, that policy is translated into a process that is automatically launched whenever a developer submits code, and code that violates the policy may be automatically rejected.

In some embodiments, SaC may be implemented by initially classifying workloads (e.g., by sensitivity, by criticality, by deployment model, by segment). Policies that can be instantiated as code may subsequently be designed. For example, compute-related policies may be designed, access-related policies may be designed, application-related policies may be designed, network-related policies may be designed, data-related policies may be designed, and so on. Security as code may then be instantiated through architecture and automation, as successful implementation of SaC can benefit from making key architectural-design decisions and executing the right automation capabilities. Next, operating model protections may be built and supported. For example, an operating model may "shift left" to maximize self-service and achieve full-life-cycle security automation (e.g., by standardizing common development toolchains, CI/CD pipelines, and the like). In such an example, security policies and access controls may be part of the pipeline, automatic code review and bug/defect detection may be performed, automated build processes may be performed, vulnerability scanning may be performed, checks against a risk-

control framework may be made, and other tasks may be performed all before deploying an infrastructure or components thereof.

The systems described herein may be useful in analyzing, monitoring, evaluating, or otherwise observing a GitOps environment. In a GitOps environment, Git may be viewed as the one and only source of truth. As such, GitOps may require that the desired state of infrastructure (e.g., a customer's cloud deployment) be stored in version control such that the entire audit trail of changes to such infrastructure can be viewed or audited. In a GitOps environment, all changes to infrastructure are embodied as fully traceable commits that are associated with committer information, commit IDs, time stamps, and/or other information. In such an embodiment, both an application and the infrastructure (e.g., a customer's cloud deployment) that supports the execution of the application are therefore versioned artifacts and can be audited using the gold standards of software development and delivery. Readers will appreciate that while the systems described herein are described as analyzing, monitoring, evaluating, or otherwise observing a GitOps environment, in other embodiments other source control mechanisms may be utilized for creating infrastructure, making changes to infrastructure, and so on. In these embodiments, the systems described herein may similarly be used for analyzing, monitoring, evaluating, or otherwise observing such environments.

As described in other portions of the present disclosure, the systems described herein may be used to analyze, monitor, evaluate, or otherwise observe a customer's cloud deployment. While securing traditional datacenters requires managing and securing an IP-based perimeter with networks and firewalls, hardware security modules ('HSMs'), security information and event management ('SIEM') technologies, and other physical access restrictions, such solutions are not particularly useful when applied to cloud deployments. As such, the systems described herein may be configured to interact with and even monitor other solutions that are appropriate for cloud deployments such as, for example, "zero trust" solutions.

A zero trust security model (a.k.a., zero trust architecture) describes an approach to the design and implementation of IT systems. A primary concept behind zero trust is that devices should not be trusted by default, even if they are connected to a managed corporate network such as the corporate LAN and even if they were previously verified. Zero trust security models help prevent successful breaches by eliminating the concept of trust from an organization's network architecture. Zero trust security models can include multiple forms of authentication and authorization (e.g., machine authentication and authorization, human/user authentication and authorization) and can also be used to control multiple types of accesses or interactions (e.g., machine-to-machine access, human-to-machine access).

In some embodiments, the systems described herein may be configured to interact with zero trust solutions in a variety of ways. For example, agents that collect input data for the systems described herein (or other components of such systems) may be configured to access various machines, applications, data sources, or other entity through a zero trust solution, especially where local instances of the systems described herein are deployed at edge locations. Likewise, given that zero trust solutions may be part of a customer's cloud deployment, the zero trust solution itself may be monitored to identify vulnerabilities, anomalies, and so on. For example, network traffic to and from the zero trust solution may be analyzed, the zero trust solution may be monitored to detect unusual interactions, log files generated by the zero trust solution may be gathered and analyzed, and so on.

In some embodiments, the systems described herein may leverage various tools and mechanisms in the process of performing its primary tasks (e.g., monitoring a cloud deployment). For

example, Linux eBPF is mechanism for writing code to be executed in the Linux kernel space. Through the usage of eBPF, user mode processes can hook into specific trace points in the kernel and access data structures and other information. For example, eBPF may be used to gather information that enables the systems described herein to attribute the utilization of networking resources or network traffic to specific processes. This may be useful in analyzing the behavior of a particular process, which may be important for observability/SIEM.

The systems described may be configured to collect security event logs (or any other type of log or similar record of activity) and telemetry in real time for threat detection, for analyzing compliance requirements, or for other purposes. In such embodiments, the systems described herein may analyze telemetry in real time (or near real time), as well as historical telemetry, to detect attacks or other activities of interest. The attacks or activities of interest may be analyzed to determine their potential severity and impact on an organization. In fact, the attacks or activities of interest may be reported, and relevant events, logs, or other information may be stored for subsequent examination.

In one embodiment, systems described herein may be configured to collect security event logs (or any other type of log or similar record of activity) and telemetry in real time to provide customers with a SIEM or SIEM-like solution. SIEM technology aggregates event data produced by security devices, network infrastructure, systems, applications, or other source. Centralizing all of the data that may be generated by a cloud deployment may be challenging for a traditional SIEM, however, as each component in a cloud deployment may generate log data or other forms of machine data, such that the collective amount of data that can be used to monitor the cloud deployment can grow to be quite large. A traditional SIEM architecture, where data is centralized and aggregated, can quickly result in large amounts of data that may be expensive to store, process, retain, and so on. As such, SIEM technologies may frequently be implemented such that silos are created to separate the data.

In some embodiments of the present disclosure, data that is ingested by the systems described herein may be stored in a cloud-based data warehouse such as those provided by Snowflake and others. Given that companies like Snowflake offer data analytics and other services to operate on data that is stored in their data warehouses, in some embodiments one or more of the components of the systems described herein may be deployed in or near Snowflake as part of a secure data lake architecture (a.k.a., a security data lake architecture, a security data lake/warehouse). In such an embodiment, components of the systems described herein may be deployed in or near Snowflake to collect data, transform data, analyze data for the purposes of detecting threats or vulnerabilities, initiate remediation workflows, generate alerts, or perform any of the other functions that can be performed by the systems described herein. In such embodiments, data may be received from a variety of sources (e.g., EDR or EDR-like tools that handle endpoint data, cloud access security broker ('CASB') or CASB-like tools that handle data describing interactions with cloud applications, Identity and Access Management ('IAM') or IAM-like tools, and many others), normalized for storage in a data warehouse, and such normalized data may be used by the systems described herein. In fact, the systems described herein may actually implement the data sources (e.g., an EDR tool, a CASB tool, an IAM tool) described above.

In some embodiments one data source that is ingested by the systems described herein is log data, although other forms of data such as network telemetry data (flows and packets) and/or many other forms of data may also be utilized. In some embodiments, event data can be combined with contextual information about users, assets, threats, vulnerabilities, and so on, for the purposes of scoring, prioritization and expediting investigations. In some embodiments, input

data may be normalized, so that events, data, contextual information, or other information from disparate sources can be analyzed more efficiently for specific purposes (e.g., network security event monitoring, user activity monitoring, compliance reporting). The embodiments described here offer real-time analysis of events for security monitoring, advanced analysis of user and entity behaviors, querying and long-range analytics for historical analysis, other support for incident investigation and management, reporting (for compliance requirements, for example), and other functionality.

In some embodiments, the systems described herein may be part of an application performance monitoring ('APM') solution. APM software and tools enable the observation of application behavior, observation of its infrastructure dependencies, observation of users and business key performance indicators ('KPIs') throughout the application's life cycle, and more. The applications being observed may be developed internally, as packaged applications, as software as a service ('SaaS'), or embodied in some other ways. In such embodiments, the systems described herein may provide one or more of the following capabilities:

The ability to operate as an analytics platform that ingests, analyzes, and builds context from traces, metrics, logs, and other sources.

Automated discovery and mapping of an application and its infrastructure components.

Observation of an application's complete transactional behavior, including interactions over a data communications network.

Monitoring of applications running on mobile (native and browser) and desktop devices.

Identification of probable root causes of an application's performance problems and their impact on business outcomes.

Integration capabilities with automation and service management tools.

Analysis of business KPIs and user journeys (for example, login to check-out).

Domain-agnostic analytics capabilities for integrating data from third-party sources.

Endpoint monitoring to understand the user experience and its impact on business outcomes.

Support for virtual desktop infrastructure ('VDI') monitoring.

In embodiments where the systems described herein are used for APM, some components of the system may be modified, other components may be added, some components may be removed, and other components may remain the same. In such an example, similar mechanisms as described elsewhere in this disclosure may be used to collect information from the applications, network resources used by the application, and so on. The graph based modelling techniques may also be leveraged to perform some of the functions mentioned above, or other functions as needed.

In some embodiments, the systems described herein may be part of a solution for developing and/or managing artificial intelligence ('AI') or machine learning ('ML') applications. For example, the systems described herein may be part of an AutoML tool that automate the tasks associated with developing and deploying ML models. In such an example, the systems described herein may perform various functions as part of an AutoML tool such as, for example, monitoring the performance of a series of processes, microservices, and so on that are used to collectively form the AutoML tool. In other embodiments, the systems described herein may perform other functions as part of an AutoML tool or may be used to monitor, analyze, or otherwise observe an environment that the AutoML tool is deployed within.

In some embodiments, the systems described herein may be used to manage, analyze, or otherwise observe deployments that include other forms of AI/ML tools. For example, the systems described herein may manage, analyze, or otherwise observe deployments that include

AI services. AI services are, like other resources in an as-a-service model, ready-made models and AI applications that are consumable as services and made available through APIs. In such an example, rather than using their own data to build and train models for common activities, organizations may access pre-trained models that accomplish specific tasks. Whether an organization needs natural language processing ('NLP'), automatic speech recognition ('ASR'), image recognition, or some other capability, AI services simply plug-and-play into an application through an API. Likewise, the systems described herein may be used to manage, analyze, or otherwise observe deployments that include other forms of AI/ML tools such as Amazon Sagemaker (or other cloud machine-learning platform that enables developers to create, train, and deploy ML models) and related services such as Data Wrangler (a service to accelerate data prep for ML) and Pipelines (a CI/CD service for ML).

In some embodiments, the systems described herein may be used to manage, analyze, or otherwise observe deployments that include various data services. For example, data services may include secure data sharing services, data marketplace services, private data exchanges services, and others. Secure data sharing services can allow access to live data from its original location, where those who are granted access to the data simply reference the data in a controlled and secure manner, without latency or contention from concurrent users. Because changes to data are made to a single version, data remains up-to-date for all consumers, which ensures data models are always using the latest version of such data. Data marketplace services operate as a single location to access live, ready-to-query data (or data that is otherwise ready for some other use). A data marketplace can even include a "feature stores," which can allow data scientists to repurpose existing work. For example, once a data scientist has converted raw data into a metric (e.g., costs of goods sold), this universal metric can be found quickly and used by other data scientists for quick analysis against that data.

In some embodiments, the systems described herein may be used to manage, analyze, or otherwise observe deployments that include distributed training engines or similar mechanisms such as, for example, such as tools built on Dask. Dask is an open source library for parallel computing that is written in Python. Dask is designed to enable data scientists to improve model accuracy faster, as Dask enables data scientists can do everything in Python end-to-end, which means that they no longer need to convert their code to execute in environments like Apache Spark. The result is reduced complexity and increased efficiency. The systems described herein may also be used to manage, analyze, or otherwise observe deployments that include technologies such as RAPIDS (an open source Python framework which is built on top of Dask). RAPIDS optimizes compute time and speed by providing data pipelines and executing data science code entirely on graphics processing units (GPUs) rather than CPUs. Multi-cluster, shared data architecture, DataFrames, Java user-defined functions (UDF) are supported to enable trained models to run within a data warehouse.

In some embodiments, the systems described herein may be leveraged for the specific use case of detecting and/or remediating ransomware attacks and/or other malicious action taken with respect to data, systems, and/or other resources associated with one or more entities. Ransomware is a type of malware from cryptovirology that threatens to publish the victim's data or perpetually block access to such data unless a ransom is paid. In such embodiments, ransomware attacks may be carried out in a manner such that patterns (e.g., specific process-to-process communications, specific data access patterns, unusual amounts of encryption/re-encryption activities) emerge, where the systems described herein may monitor for such patterns. Alternatively, ransomware attacks may involve behavior that deviates from normal behavior of a

cloud deployment that is not experiencing a ransomware attack, such that the mere presence of unusual activity may trigger the systems described herein to generate alerts or take some other action, even without explicit knowledge that the unusual activity is associated with a ransomware attack.

In some embodiments, particular policies may be put in place the systems described herein may be configured to enforce such policies as part of an effort to thwart ransomware attacks. For example, particular network sharing protocols (e.g., Common Internet File System ('CIFS'), Network File System ('NFS')) may be avoided when implementing storage for backup data, policies that protect backup systems may be implemented and enforced to ensure that usable backups are always available, multifactor authentication for particular accounts may be utilized and accounts may be configured with the minimum privilege required to function, isolated recovery environments may be created and isolation may be monitored and enforced to ensure the integrity of the recovery environment, and so on. As described in the present disclosure, the systems described herein may be configured to explicitly enforce such policies or may be configured to detect unusual activity that represents a violation of such policies, such that the mere presence of unusual activity may trigger the systems described herein to generate alerts or take some other action, even without explicit knowledge that the unusual activity is associated with a violation of a particular policy.

Readers will appreciate that ransomware attacks are often deployed as part of a larger attack that may involve, for example:

Penetration of the network through means such as, for example, stolen credentials and remote access malware.

Stealing of credentials for critical system accounts, including subverting critical administrative accounts that control systems such as backup, Active Directory ('AD'), DNS, storage admin consoles, and/or other key systems.

Attacks on a backup administration console to turn off or modify backup jobs, change retention policies, or even provide a roadmap to where sensitive application data is stored. Data theft attacks.

As a result of the many aspects that are part of a ransomware attack, embodiments of the present disclosure may be configured as follows:

The systems may include one or more components that detect malicious activity based on the behavior of a process.

The systems may include one or more components that store indicator of compromise ('IOC') or indicator of attack ('IOA') data for retrospective analysis.

The systems may include one or more components that detect and block fileless malware attacks.

The systems may include one or more components that remove malware automatically when detected.

The systems may include a cloud-based, SaaS-style, multitenant infrastructure.

The systems may include one or more components that identify changes made by malware and provide the recommended remediation steps or a rollback capability.

The systems may include one or more components that detect various application vulnerabilities and memory exploit techniques.

The systems may include one or more components that continue to collect suspicious event data even when a managed endpoint is outside of an organization's network.

The systems may include one or more components that perform static, on-demand malware detection scans of folders, drives, devices, or other entities.

The systems may include data loss prevention (DLP) functionality.

In some embodiments, the systems described herein may manage, analyze, or otherwise observe deployments that include deception technologies. Deception technologies allow for the use of decoys that may be generated based on scans of true network areas and data. Such decoys may be deployed as mock networks running on the same infrastructure as the real networks, but when an intruder attempts to enter the real network, they are directed to the false network and security is immediately notified. Such technologies may be useful for detecting and stopping various types of cyber threats such as, for example, Advanced Persistent Threats ('APTs'), malware, ransomware, credential dumping, lateral movement and malicious insiders. To continue to outsmart increasingly sophisticated attackers, these solutions may continuously deploy, support, refresh and respond to deception alerts.

In some embodiments, the systems described herein may manage, analyze, or otherwise observe deployments that include various authentication technologies, such as multi-factor authentication and role-based authentication. In fact, the authentication technologies may be included in the set of resources that are managed, analyzed, or otherwise observed as interactions with the authentication technologies may monitored. Likewise, log files or other information retained by the authentication technologies may be gathered by one or more agents and used as input to the systems described herein.

In some embodiments, the systems described herein may be leveraged for the specific use case of detecting supply chain attacks. More specifically, the systems described herein may be used to monitor a deployment that includes software components, virtualized hardware components, and other components of an organization's supply chain such that interactions with an outside partner or provider with access to an organization's systems and data can be monitored. In such embodiments, supply chain attacks may be carried out in a manner such that patterns (e.g., specific interactions between internal and external systems) emerge, where the systems described herein may monitor for such patterns. Alternatively, supply chain attacks may involve behavior that deviates from normal behavior of a cloud deployment that is not experiencing a supply chain attack, such that the mere presence of unusual activity may trigger the systems described herein to generate alerts or take some other action, even without explicit knowledge that the unusual activity is associated with a supply chain attack.

In some embodiments, the systems described herein may be leveraged for other specific use cases such as, for example, detecting the presence of (or preventing infiltration from) cryptocurrency miners (e.g., bitcoin miners), token miners, hashing activity, non-fungible token activity, other viruses, other malware, and so on. As described in the present disclosure, the systems described herein may monitor for such threats using known patterns or by detecting unusual activity, such that the mere presence of unusual activity may trigger the systems described herein to generate alerts or take some other action, even without explicit knowledge that the unusual activity is associated with a particular type of threat, intrusion, vulnerability, and so on.

The systems described herein may also be leveraged for endpoint protection, such the systems described herein form all of or part of an endpoint protection platform. In such an embodiment, agents, sensors, or similar mechanisms may be deployed on or near managed endpoints such as computers, servers, virtualized hardware, internet of things ('IotT') devices, mobile devices, phones, tablets, watches, other personal digital devices, storage devices, thumb drives, secure data storage cards, or some other entity. In such an example, the endpoint protection platform may provide functionality such as:

Prevention and protection against security threats including malware that uses file-based and fileless exploits.
The ability to apply control (allow/block) to access of software, scripts, processes, microservices, and so on.
The ability to detect and prevent threats using behavioral analysis of device activity, application activity, user activity, and/or other data.
The ability for facilities to investigate incidents further and/or obtain guidance for remediation when exploits evade protection controls
The ability to collect and report on inventory, configuration and policy management of the endpoints.
The ability to manage and report on operating system security control status for the monitored endpoints.
The ability to scan systems for vulnerabilities and report/manage the installation of security patches.
The ability to report on internet, network and/or application activity to derive additional indications of potentially malicious activity.
Example embodiments are described in which policy enforcement, threat detection, or some other function is carried out by the systems described herein by detecting unusual activity, such that the mere presence of unusual activity may trigger the systems described herein to generate alerts or take some other action, even without explicit knowledge that the unusual activity is associated with a particular type of threat, intrusion, vulnerability, and so on. Although these examples are largely described in terms of identifying unusual activity, in these examples the systems described herein may be configured to learn what constitutes 'normal activity'—where 'normal activity' is activity observed, modeled, or otherwise identified in the absence of a particular type of threat, intrusion, vulnerability, and so on. As such, detecting 'unusual activity' may alternatively be viewed as detecting a deviation from 'normal activity' such that 'unusual activity' does not need to be identified and sought out. Instead, deviations from 'normal activity' may be assumed to be 'unusual activity'.
Readers will appreciate that while specific examples of the functionality that the systems described herein can provide are included in the present disclosure, such examples are not to be interpreted as limitations as to the functionality that the systems described herein can provide. Other functionality may be provided by the systems described herein, all of which are within the scope of the present disclosure. For the purposes of illustration and not as a limitation, additional examples can include governance, risk, and compliance ('GRC'), threat detection and incident response, identity and access management, network and infrastructure security, data protection and privacy, identity and access management ('IAM'), and many others.
In order to provide the functionality described above, the systems described herein or the deployments that are monitored by such systems may implement a variety of techniques. For example, the systems described herein or the deployments that are monitored by such systems may tag data and logs to provide meaning or context, persistent monitoring techniques may be used to monitor a deployment at all times and in real time, custom alerts may be generated based on rules, tags, and/or known baselines from one or more polygraphs, and so on.
Although examples are described above where data may be collected from one or more agents, in some embodiments other methods and mechanisms for obtaining data may be utilized. For example, some embodiments may utilize agentless deployments where no agent (or similar mechanism) is deployed on one or more customer devices, deployed within a customer's cloud

deployment, or deployed at another location that is external to the data platform. In such embodiments, the data platform may acquire data through one or more APIs such as the APIs that are available through various cloud services. For example, one or more APIs that enable a user to access data captured by Amazon CloudTrail may be utilized by the data platform to obtain data from a customer's cloud deployment without the use of an agent that is deployed on the customer's resources. In some embodiments, agents may be deployed as part of a data acquisition service or tool that does not utilize a customer's resources or environment. In some embodiments, agents (deployed on a customer's resources or elsewhere) and mechanisms in the data platform that can be used to obtain data from through one or more APIs such as the APIs that are available through various cloud services may be utilized. In some embodiments, one or more cloud services themselves may be configured to push data to some entity (deployed anywhere), which may or may not be an agent. In some embodiments, other data acquisition techniques may be utilized, including combinations and variations of the techniques described above, each of which is within the scope of the present disclosure.

Readers will appreciate that while specific examples of the cloud deployments that may be monitored, analyzed, or otherwise observed by the systems described herein have been provided, such examples are not to be interpreted as limitations as to the types of deployments that may be monitored, analyzed, or otherwise observed by the systems described herein. Other deployments may be monitored, analyzed, or otherwise observed by the systems described herein, all of which are within the scope of the present disclosure. For the purposes of illustration and not as a limitation, additional examples can include multi-cloud deployments, on-premises environments, hybrid cloud environments, sovereign cloud environments, heterogeneous environments, DevOps environments, DevSecOps environments, GitOps environments, quantum computing environments, data fabrics, composable applications, composable networks, decentralized applications, and many others.

Readers will appreciate that while specific examples of the types of data that may be collected, transformed, stored, and/or analyzed by the systems described herein have been provided, such examples are not to be interpreted as limitations as to the types of data that may be collected, transformed, stored, and/or analyzed by the systems described herein. Other types of data can include, for example, data collected from different tools (e.g., DevOps tools, DevSecOps, GitOps tools), different forms of network data (e.g., routing data, network translation data, message payload data, Wi-Fi data, Bluetooth data, personal area networking data, payment device data, near field communication data, metadata describing interactions carried out over a network, and many others), data describing processes executing in a container, lambda, EC2 instance, virtual machine, or other execution environment), information describing the execution environment itself, and many other types of data.

For further explanation, FIG. **5**A sets forth a system for providing many of the features described herein for user devices as a distributed edge service in accordance with some embodiments of the present disclosure. The system depicted in FIG. **5**A includes a distributed edge platform **510**. The distributed edge platform **510** may be similar to the systems described above where the distributed edge platform **510** can be used to perform tasks such as, for example, anomaly detection, threat detection, vulnerability detection, compliance monitoring, and many others. The distributed edge platform **510** may be deployed in a distributed fashion, such that instances of the distributed edge platform **510** are deployed on geographically distributed execution environments, as will be described in greater detail below.

The distributed edge platform **510** depicted in FIG. **5**A may be utilized to provide for continuous risk behavior based security to user devices. Such security is 'continuous' in the sense that data regarding the activity of a user device is continuously gathered and evaluated in real-time (or near real-time) rather than performing batch-based evaluation. Such security is 'behavior based' in the sense that various behaviors of the user device are used as the primary inputs into security evaluations. In many cases, such behaviors may not be concerning on their own, but instead may represent a deviation from typical device activity that warrants additional investigation. For example, if the user device is connecting to a cloud deployment, creating EC2 instances, and executing software on those EC2 instances, those steps alone may not be concerning. If this happens at hours when the user is known to be asleep and the user has never logged into the cloud deployment nor deployed software on EC2 instances in the cloud, however, this behavior may be deemed to be suspicious.

The example depicted in FIG. **5**A includes a private environment **502** that is supporting a plurality of resources **504** *a-n*. The private environment **502** may be embodied as a customer's datacenter, as a virtual private cloud for a particular customer, as co-located resources, or in some other way. The private environment **502** may include a collection of hardware resources, software resources, networking resources, and other resources so that the private environment **502** can be used to provide an organization or some other entity with an environment to execute their secure applications, store their secure data, and so on. In this example, the private environment **502** is 'private' in the sense that it is not available for public consumption. The private environment **502** depicted in FIG. **5**A is supporting a plurality of resources **504** *a-n* that may include, for example, software applications, databases, file systems, various services (e.g., whether local SaaS offerings), development tools (e.g., automation servers, code repositories, etc. . . .), and so on.

The private environment **502** also includes a connector **506** that can be used to connect the private environment to the distributed edge platform **510** via a zero trust **508** authentication service. The connecter **506** may be embodied, for example, as one or more modules of computer program instructions executing on computer hardware, virtualized hardware, or in some other execution environment (including on resources in a dedicated networking components such as a router). The connector may be configured to act as a communications interface between the private environment **502** and the distributed edge platform **510**. In such an example, access between the private environment **502** and the distributed edge platform **510** may involve the zero trust **508** authentication service, as any participants in data communications between the private environment **502**, the distributed edge platform **510**, and the user devices **512**, **514**, **522** that are monitored by the distributed edge platform **510** must be authenticated by the zero trust **508** authentication service. The zero trust **508** authentication service may be embodied, for example, as a third party authentication service such as Okta, or in some other way. In fact, readers will appreciate that the distributed edge platform **510** may be integrated with (and may leverage) a variety of third party tools such as identity and access management tools, Mobile device management ('MDM') tools, and so on.

In the example depicted in FIG. **5**A, a user device **512** can access the resources **504** *a-n* in the private environment **502** via the zero trust **508** authentication service. In such an example, because all access occurs through the zero trust **508** authentication service, the user device **512** need not connect to a VPN, go through a firewall, or use a similar mechanism in order to securely access the resources **504** *a-n* in the private environment **502**. In fact, the distributed edge platform **510** may implement policies that identify which users may access

which resources **504** *a-n*, what privileges each user has, and other policies so that the user device **512** is only routed to (and given access to) a particular resource **504** *a-n* in the private environment **502** if the policies allow for such access. The distributed edge platform **510** can do all sorts of behavior analysis (i.e., analysis of device activity and user activity as described below) to add in anomaly detection capabilities, threat assessment, risk assessment, and other security related capabilities as described elsewhere in the present disclosure.

The example depicted in FIG. **5**A also includes a SaaS environment **518** that is supporting a plurality of resources **520** *a-n*. The SaaS environment **518** may be embodied, for example, as a public cloud that is accessible using a particular account, as an environment provided by the vendor of some SaaS offering, or in some other way. The SaaS environment **518** may include a collection of hardware resources, software resources, networking resources, and other resources so that the SaaS environment **518** can be used to provide a vendor of software that is consumed as-a-service to offer their SaaS products. The SaaS offerings can include any software offered as-a-service including, for example, Salesforce, Office365, and many others. In this example, the resources **520** *a-n* that may include software applications, databases, file systems, or anything else offered as-a-service.

In the example depicted in FIG. **5**A, a user device **514** can access the resources **520** *a-n* in the SaaS environment **518** via a SaaS security **516** module. The SaaS security **516** module may be embodied, for example, as a login service or similar service that is implemented by a cloud computing environment or service vendor to control access to one or more SaaS offerings. In such a way, access to the SaaS environment **518** may only occur in accordance with the requirements put in place by the cloud computing environment, the service vendor, or similar entity that controls access to one or more SaaS offerings.

The example depicted in FIG. **5**A also includes a public internet **528** that is supporting a plurality of resources **526** *a-n*. The public internet **528** may include a collection of hardware resources, software resources, networking resources, and other resources that can be used to offer resources **526** *a-n* such as websites, social media platforms, and anything else publicly accessible via a web browser, mobile application, or other appropriate interface.

In the example depicted in FIG. **5**A, a user device **522** can access the resources **526** *a-n* in the public internet **528** via a network security **524** module. The network security **524** module may be embodied, for example, as one or more modules of computer program instructions that are configured to perform tasks such as SSL inspection, DNS inspection, and other tasks described in the present disclosure. In such an example, the network security module may be configured to protect the user device **522** from malware intrusions, computer viruses, or other threats that can originate from the public internet **528**.

The distributed edge platform **510** may implement policies that identify which users may access which resources **520** *a-n*, what privileges each user has, and other policies so that the user device **514** is only able to access various resources if the policies allow for such access. In fact, the distributed edge platform **510** can do all sorts of behavior analysis (i.e., analysis of device activity and user activity as described below) to add in anomaly detection capabilities, threat assessment, risk assessment, and other security related capabilities as described elsewhere in the present disclosure.

The distributed edge platform **510** depicted in FIG. **5**A includes a real-time visibility and policy enforcement **530** module. The real-time visibility and policy enforcement **530** module may be embodied, for example, as one or more modules of computer program instructions executing on computer hardware, virtualized hardware, or in some other execution environment. The real-time

visibility and policy enforcement **530** module may be configured to carry out many of the steps described above such as, for example, monitoring user activity (e.g., via data communications involving a user device or in some other way) to enforce various policies describing how the user devices may be utilized, what resources the user devices may access, what privileges the user device has, and so on. The real-time visibility and policy enforcement **530** module may also enable an administrator to have visibility into the activities associated with the monitored user devices. The real-time visibility and policy enforcement **530** module may be used to populate a monitoring interface used by the administrator, the real-time visibility and policy enforcement **530** module may enable the administrator to query the distributed edge platform **510** for information related to user activity or device activity, and so on.

The distributed edge platform **510** depicted in FIG. **5**A also includes a user behavior anomaly detection **532** module. The user behavior anomaly detection **532** module may be embodied, for example, as one or more modules of computer program instructions executing on computer hardware, virtualized hardware, or in some other execution environment. The user behavior anomaly detection **532** may be configured to analyze user behavior, user device activity, or other information to detect anomalous behavior as described in greater detail elsewhere in the present disclosure.

The distributed edge platform **510** depicted in FIG. **5**A also includes an events, workflows, and auto management **534** module. The events, workflows, and auto management **534** module may be embodied, for example, as one or more modules of computer program instructions executing on computer hardware, virtualized hardware, or in some other execution environment. The events, workflows, and auto management **534** module may be configured to generate alerts, initiate a remediation workflow (or some other workflow), or perform other automatic remediation tasks as described in greater detail elsewhere in the present disclosure.

For further explanation, FIG. **5**B sets forth a system for providing many of the features described herein for user devices as a distributed edge service in accordance with some embodiments of the present disclosure. The example depicted in FIG. **5**B illustrates the distributed nature of the distributed edge service. The example depicted in FIG. **5**B illustrates instances of the distributed edge platform **510** *a*, **510** *b*, **510** *c*, **510** *d* that are executing at distinct population centers **540** *a*, **540** *b*, **540** *c*, **540** *d*. Each of the population centers **540** *a*, **540** *b*, **540** *c*, **540** *d* may be embodied, for example, as a geographically distributed execution environment such as a distinct availability zone that is provided by a cloud services provider such as AWS, GCP, Azure, or others. In fact, the geographically distributed execution environments (represented here as population centers **540** *a*, **540** *b*, **540** *c*, **540** *d*) may even be provided by multiple cloud services providers (e.g., population center **540** *a* is supported by AWS and population center **540** *b* is supported by GCP).

Readers will appreciate that each instance of the distributed edge platform **510** *a*, **510** *b*, **510** *c*, **510** *d* may, in some embodiments, be deployed on a set of appliances that are deployed in various locations. Each appliance may include, for example, one or more servers or other computing devices, one or more networking devices, one or more storage devices, and so on. In such an example, each appliance may be configured to execute a particular instance of the distributed edge platform **510** *a*, **510** *b*, **510** *c*, **510** *d*.

In the example depicted in FIG. **5**B, the instance of the distributed edge platform **510** *a*, **510** *b*, **510** *c*, **510** *d* that is accessed by a particular user device **512**, **514**, **522**, **542** may be dependent upon the relative proximity between each population center **540** *a*, **540** *b*, **540** *c*, **540** *d* and each user device **512**, **514**, **522**, **542**. For

example, each user device **512**, **514**, **522**, **542** may connect to the instance of the distributed edge platform **510** *a*, **510** *b*, **510** *c*, **510** *d* that is supported by the population center **540** *a*, **540** *b*, **540** *c*, **540** *d* that is most physically proximate to the user device **512**, **514**, **522**, **542**. The various instances of the distributed edge platform **510** *a*, **510** *b*, **510** *c*, **510** *d* may be updated in a coordinated fashion and may share access to the same information such that each instance operates in the same manner as any other instance, even if two instances are deployed in different ways (e.g., on different underlying resources).

For further explanation, FIG. **6** sets forth a flow chart illustrating an example method of detecting deviations from typical user behavior in accordance with some embodiments of the present disclosure. As will be described in greater detail below, detecting deviations from typical user behavior can be carried out with respect to end user devices. For example, detecting deviations from typical user behavior can include detecting typical user behavior as reflected by the manner in which a particular device (e.g., a laptop, a smartphone) is typically used and identifying situations in which the particular device is being used in a manner that deviates from the typical usage pattern. Although not expressly illustrated in FIG. **6** , the methods described in FIG. **6** and elsewhere in the present disclosure may be carried out by one or more modules of computer program instructions executing on computer hardware, virtualized computer hardware, containers, or in some other execution environment.

The example method depicted in FIG. **6** includes identifying **602** a location of a device that is associated with a user. The device that is associated with a user may be embodied, for example, as a smartphone, as a tablet computer, as a laptop computer, or as some other device. The device may be associated with the user because the user is logged into the device, the device has been designated for use by the user (e.g., the device is a company issued laptop provided by the user's employer), or the device is otherwise associated with the user. The location of the device may be embodied, for example, as a city and state (e.g., Los Angeles, CA), as a label of a known location (e.g., 'home'), or designated in some other way.

Identifying **602** a location of a device that is associated with a user may be carried out in a variety of ways. In fact, identifying **602** a location of the device may be carried out using any of multiple mechanisms for determining a device's location and (in some embodiments) verifying that each mechanism yields the same outcome, or at least the same outcome within a predetermined threshold. For example, identifying **602** a location of the device may be carried out by identifying wireless networks or wireless access points that the device can connect to and determining the location of the available wireless networks or wireless access points. Once the locations of wireless networks or wireless access points that the device can connect to have been identified, other mechanisms may be used to verify the location of the device. For example, the source IP address of a data communications packet generated by the device may be used to determine the location of the device. In such an example, if the locations of wireless networks or wireless access points that the device can connect to and the location of the source IP address of a data communications packet generated by the device match each other, the matching locations may be identified **602** as being the location of a device. The mechanisms for determining a device's location will be explained in greater detail below.

The example method depicted in FIG. **6** also includes determining **604** device activity associated with the user. The device activity associated with the user can include information describing the usage of the device. The device activity can include, for example, information describing applications that are being executed or utilized on the device, information describing how those

applications are being utilized, information describing files being accessed via the device, information describing data sources being accessed via the device, information describing data communications coming into and flowing out of the device, and many others. In general, the device activity can be used to ascertain how a device is being used, when the device is being used, where the device is being used, or any other quantifiable aspect of device usage.

Determining **604** device activity associated with the user may be carried out, for example, by one or more data collection agents that are executing on the device, by one or more data collection agents that executing at some other location off of the device, or in some other way. That is, while some information describing device activity may come from the device (OS version, what apps are running locally, etc.), the information describing device activity may also come from outside the device. For example, other cloud services that report what users are doing may be leveraged (e.g., activity logs from Office **365** may be used, activity logs from Dropbox may be used). In such an example, even if no agents or other data collection programs were deployed on the device itself, information describing device activity may still be obtained. As such, determining **604** device activity associated with the user may be carried out by local agents executing on the monitored device itself, by agents executing elsewhere, or by combinations and variations thereof. The data collection agents (whether deployed on the user's device or elsewhere) may include, for example, one or more programs that can carry out a stream-based capture of network traffic in and out of the device (i.e., stream-oriented traffic processing), one or more programs that capture the usage of a device interface (e.g., a keystroke recorder, a recording program for capturing data acquired via a microphone), or other programs. In such a way, the agents may capture various aspects describing how the device is being used, when it is being used, by whom it is being used, where the device is located when being used, and so on.

The example method depicted in FIG. **6** also includes determining **606**, based on a profile associated with the user, that the device activity associated with the user deviates from normal activity for the user. The profile associated with the user may include information describing normal activity for the user. The normal activity for the user may be determined, for example, based on historical usage of the device (or some other device) associated with the user. That is, normal activity may be learned through an analysis of how the device has historically been used rather than being specified exclusively as a set of rules. The normal activity may include, for example, an identification of applications on the device that are accessed by the user, the times that those applications are accessed, the locations from which those applications are accessed, the order in which the applications on the device are typically accessed by the user, and so on. The particular mechanics of creating a user profile and learning what is normal activity for the device will be described in greater detail below.

In the example method depicted in FIG. **6** , determining **606**, based on a profile associated with the user, that the device activity associated with the user deviates from normal activity for the user may be carried out by comparing the device activity with the profile associated with the user. A comparison between the device activity and the profile associated with the user may reveal that device activity does not align with the profile associated with the user, which may be treated as a detection of abnormal activity.

Readers will appreciate that comparisons the device activity and the profile associated with the user may utilize ranges, thresholds, or similar concepts to allow for minor deviations between the device activity and the profile associated with the user. For example, if the profile associated with the user indicates that the device is typically located at the user's office between the hours of 9 AM and 5 PM on weekdays, but the device activity reveals that the user is still at the office at 6

PM on a particular Tuesday evening, this minor deviation may be tolerated and may not rise to the level of triggering an alarm. In contrast, if the device activity reveals that the user is at the office at 2:30 AM on a Sunday morning, this may be viewed as being a larger deviation. In this example, a threshold may be utilized such that the user deviating from their normal activity by 90 minutes (as a very simplified example used for ease of explanation) does not rise to the level of abnormal activity that would trigger an alarm or cause some other remediation workflow to be initiated. In other embodiments, other mechanisms may be used to allow for minor deviations between the device activity and the profile associated with the user, where such minor deviations do not result in determining **606** that the device activity associated with the user deviates from normal activity for the user.

For further explanation, FIG. **7** sets forth a flow chart illustrating an additional example method of detecting deviations from typical user behavior in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **7** is similar to the example method depicted in FIG. **6** , as the example method depicted in FIG. **7** also includes identifying **602** a location of a device that is associated with a user, determining **604** device activity associated with the user, and determining **606** that the device activity associated with the user deviates from normal activity for the user.

The example method depicted in FIG. **7** also includes generating **702** the profile associated with the user. Generating **702** the profile associated with the user may be carried out, for example, by utilizing data gathered by the agents described above as input to one or more machine learning algorithms. In such a way, patterns may be identified and correlations may be detected that represent the normal activity of the user. Such machine learning algorithms may detect, for example, that a particular application is normally only used at certain times and from certain locations. For example, the data gathered by the agents described above may reveal that a VPN client on the device is typically only used during traditional business hours (e.g., 8 AM-6 PM) on weekdays and when the device is located at a location other than the user's office (as the private network may be directly accessible when the device is being used at the user's office). Readers will appreciate that in other embodiments, the profile associated with the user may be generated **702** in some other way. In such an example, the profile associated with the user may be expressed as a trained machine learning model that may be deployed to differentiate between normal activity and abnormal activity associated with a user (or a user device).

In the example method depicted in FIG. **7** , determining **604** device activity associated with the user can include identifying **704** one or more applications being accessed on the device. Identifying **704** one or more applications being accessed on the device may be carried out, for example, by an agent that is executing on the device. For example, the agent may identify all running processes on the device and map (by the agent or by some other entity) the running processes to particular applications, the agent may identify all binaries that are loaded on the device and map (by the agent or by some other entity) the binaries to particular applications, the agent may query a device management application, or the agent may identify **704** one or more applications accessed by the device in some other way. In other embodiments, especially those in which the applications are SaaS offerings that are executed externally to the device (e.g., in a cloud environment), identifying **704** one or more applications being accessed on the device may be carried out in other ways such as, for example, by examining activity logs generated by the creator of the SaaS offering, by monitoring the device for calls to the application, through the use of an agent that monitors traffic in and out of the SaaS offering, or in some other way.

In the example method depicted in FIG. **7** , determining **604** device activity associated with the user can also include identifying **706** application behavior for the one or more applications. Identifying **706** application behavior for the one or more applications can include, for example, identifying the data communications endpoints that the applications have communicated with, identifying data that has been accessed by the applications, identifying users or accounts that have utilized the applications, identifying the time that the application was used, and so on. Such application behavior can include any quantifiable aspect describing how the application was used, when the application was used, by whom the application was used, and so on.

The example method depicted in FIG. **7** also includes, responsive to determining that the device activity associated with the user deviates from normal activity for the user, generating **708** an alert. The alert that is generated **708** may include contextual information including the specific actions (e.g., accessing a VPN client on a during non-business hours on a weekend from a previously unknown location) that caused the alert to be generated. In such an example, the alert may be issued to the user of the device, issued to a system administrator or similar entity that oversees a company's deployment, or issued elsewhere. In fact, the alert may be presented as part of a user-specific polygraph as will be described in greater detail herein.

For further explanation, FIG. **8** sets forth a flow chart illustrating an additional example method of detecting deviations from typical user behavior in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **8** is similar to the examples method depicted in FIGS. **6** and **7** , as the example method depicted in FIG. **8** also includes identifying **602** a location of a device that is associated with a user, determining **604** device activity associated with the user, and determining **606** that the device activity associated with the user deviates from normal activity for the user.

The example method depicted in FIG. **8** also includes generating **802** a user-specific polygraph. The user-specific polygraph may be generated **802** in a manner that is similar to the manner in which polygraphs are created as described above. Although the user-specific polygraphs may be similar to the polygraphs described above, the user-specific polygraph may be distinct by virtue of each entity in the user-specific polygraph being specific to a particular user or user device. As one example of a user specific polygraph, FIGS. **15**A and **15**B include examples of a user-specific polygraph. As is depicted in greater detail below, the user-specific polygraph can include the geographic location of the device and the device activity associated with the user, among other possible entities. In some embodiments, the user-specific polygraph can also include one or more alerts, as described in greater detail below.

In the examples described herein, the device activity associated with the user may be continuously monitored for deviation from normal activity. The device activity may be continuously monitored for deviation from normal activity, for example, by determining whether device activity associated with the user deviates from normal activity for the user (as specified in a user profile) every time that a change to the device activity associated with the user occurs, as part of a process that is always executing on some computing resources, or in some other way. In such a way, the systems described herein may be provide real-time or near real-time detection of a deviation from normal activity-rather than batch processing or other form of delayed processing of device activity to determine whether such activity deviates from normal activity.

In the examples described herein, the device activity associated with the user and the profile associated with the user include temporal information. The temporal information may be embodied, for example, as specific dates or times when some activity occurred or normally occurs, as relative times when some activity occurred or normally occurs, ranges of times when

some activity occurred or normally occurs, as durations of time when some activity occurred or normally occurs, and so on. As such, the time at which some activity occurred may factor into an evaluation as to whether the activity represents normal activity.

For further explanation, FIG. **9** sets forth a flow chart illustrating an example method of establishing a location profile for a user device in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **9** includes gathering **902** information associated with the location of a user device. Gathering **902** information associated with the location of a user device may be carried out not only by gathering geolocation information, but also gathering information that may be useful in determining the nature of a particular location. For example, information may be gathered **902** to determine whether the location of the user device is a location where the user device is frequently utilized, information may be gathered **902** to determine whether the location of the user device is a location where utilizing the user device to perform certain functions is allowed or prohibited, and so on. As such, the information associated with the location of a user device may be gathered **902** not just for the purposes of determining where the device would be located on a map, but to use the location of the device as an input to determining whether a user or a user device is exhibiting anomalous behavior.

Readers will appreciate that many forms of information associated with the location of a user device may gathered **902** through a variety of mechanisms. Gathering **902** information associated with the location of a user device may be carried out, for example, using location-related capabilities of the user device such as a global positioning system ('GPS') receiver, an Assisted GPS ('AGPS') chip, and so on. In other embodiments, gathering **902** information associated with the location of a user device may be carried out using data communications related capabilities of the user device. For example, a Wi-Fi adapter in the user device may detect nearby networks and the Service Set Identifier ('SSID') associated with a detected network may be mapped to a geolocation through the use of tools such as Google Geolocation API. Likewise, examining data communications traffic sent by the user device and extracting the client IP address that is associated with the data communications traffic may be used in gathering **902** information associated with the location of the user device. In such an example, the client IP address may be used to query one or more of a variety of services that convert IP addresses to geolocation information (e.g., a city/state, latitude and longitude, and so on). In fact, other embodiments could leverage image sensors or other capabilities of the user device to gather **902** information that may be informative for the purposes of identify a user device's location.

Readers will appreciate that because the information associated with the location of a user device may be gathered **902** for reasons that extend well beyond the context of a geolocation, the information associated with the location of a user device may be information that is more useful in describing the nature of a location rather than an absolute physical location. For example, the particular set of device types that the user device can communicate with may be indicative of a location. Consider an example in which the user device can detect the presence of a thermostat, a smart refrigerator, multiple smart TVs, a router, and the entertainment system of an automobile via its Wi-Fi adapter or Bluetooth adapter. In such an example, this combination of reachable device types may be taken as an indication that the user device is at a private residence. In an example where the user device can only detect the presence of other personal communications devices and also detect the presence of a wireless network that includes a phrase such as "free," "public," or "starbucks" in its network name, this combination of reachable devices may be taken as an indication that the user device is at a public location (perhaps a coffee shop). In these

examples, rather than attempting to determine a geolocation, information may be gathered **902** that can be used to determine a relative location, a type of location, characteristics of a location, or some other information that may be associated with a particular location.

The example method depicted in FIG. **9** also includes determining **904**, based on the information associated with the location of a user device, whether the user device is being accessed at a known location. A 'known' location, as the term is used here, can refer to a location that is associated with an expected profile of device utilization based on previously observing how the user device (or some other device such as a similar device, a device that is associated with similar users, and so on) is utilized at the 'known' location. For example, a user device may be expected to be used in one way when the user device is located at the user's home, the user device may be expected to be used in another way when the user device is located at the user's office, the user device may be expected to be used in yet another way when the user device is located in the user's automobile, and so on. As such, in some embodiments a location may only be determined **904** to be a 'known location' if the location has an associated set of user behaviors that would be expected to be observed when the user device is located at the 'known location'.

Readers will appreciate that a location is not necessarily 'known' in the sense that the device's geolocation is known, although in some embodiments a known location may include a specific geolocation (e.g., 123 Avenue A, San Francisco, CA). For example, while it may not be possible to determine the mailing address or street address where the user device is located, the detected presence of a particular set of other devices may be sufficient to determine that the user device is at a 'home' location, even if the exact street address of the home cannot be determined.

Likewise, detecting that the user device is located at a particular set of GPS coordinates may be insufficient for determining that the location of the user device is 'known' if the set of GPS coordinates has no known relationship to a location where the user device has previously been used or is otherwise associated with a set of behaviors that would be expected to be observed when the user device is at the GPS coordinates. In fact, in some embodiments, the location of the user device may only be determined to be a 'known' location if the user device has previously been accessed at the location or at some other location where utilization of the user device would be expected to be similar. For example, if the location of the user device is at a hotel, this location may be determined to be a 'known' location by virtue of the fact that the user device has been used in the past at other hotels (even if the user device has not been previously used at the exact hotel that it is now located at).

The example method depicted in FIG. **9** also includes, responsive to affirmatively **906** determining that the user device is being accessed at a known location, determining **908** a characterization of the known location. A 'characterization' of the known location may be embodied as description of the location that can be associated with some set of behaviors that would be expected from the user device or the user by virtue of the user device being located at a location that is characterized in a certain way. Consider an example in which the characterization is a 'home' location. In this example, a set of expected behaviors may be associated with the user or the user device by virtue of the user device being at home. For example, it may be expected that the user device accesses streaming media services (e.g., Netflix) while the user device is at home, it may be expected that the user device accesses the public internet while the user device is at home, and so on. Alternatively, consider an example in which the characterization is a 'work' location. In this example, a set of expected behaviors may be associated with the user or the user device by virtue of the user device being at home. For example, it may be expected that the user device accesses their employer's internal bill payment

systems, it may be expected that the user device accesses their employer's code repository and internal networks while at work, and so on.

In the example depicted in FIG. **9** , various 'characterizations' of locations may be generated through the use of one or more machine learning models. Machine learning models may be used, for example, to detect clusters (e.g., clusters of devices that can be reached via relatively short distance communications adapters in the devices) that can be identified as being at least a logical location. Such machine learning models may take data gathered by one or more devices to identify clusters that correspond to some location. For example, if information is gathered indicating that a collection of user devices are located in a relatively small area and that a large percentage of those devices are connected to an internal corporate network for company ABC, the machine learning models may learn that the locations each device (and possibly some corresponding area surrounding each device) can be characterized as being an office for company ABC. Readers will appreciate that other 'characterizations' may be generated using a variety of information, machine learning techniques, other labelling techniques, or generated in some other way (e.g., by asking the user of the device or some administrator of the device to characterize their current location through some interface).

Determining **908** a characterization of the known location may be carried, for example, through the use of a table, database, or some other data structure/repository that associates known locations with characterizations of each known location. Such a repository can be constructed over time by monitoring the behavior of the user device and other devices. As part of a process of generating characterizations of various locations, the activity of the user device (and other devices) may also be monitored to learn what behavior constitutes 'normal' behavior for the user or user device in various locations, using techniques such as those described herein.

The example method depicted in FIG. **9** also includes determining **910**, based on the characterization of the known location, whether device utilization is anomalous.

Determining **910** whether device utilization is anomalous may be carried out as described above, generally by determining the extent to which device activity is consistent with normal activity that would be expected to be observed when the user device is at a known location that is characterized in a particular way. Readers will appreciate that a determination so to whether device utilization is anomalous may only be based in part on the characterization of the location, as other information may also be taken into consideration (e.g., information from other devices, specific details around how the device is being used, policies that restrict how a device should be utilized at certain locations, etc. . . .).

The example method depicted in FIG. **9** also includes, responsive to determining that the user device is not 912 being accessed at a known location, determining **914** a characterization of the unknown location. Determining **914** a characterization of the unknown location may be carried out, for example, by performing some workflow to attempt to determine whether the unknown location is similar to a known characterization. For example, one characterization for various locations may be an 'in transit to work' (or something similar) characterization that is generally associated with locations between a user's home location and a user's work location, so long as there is some indication that the user device is moving at an appropriate rate from the user's home location and the user's work location. In such an example, if the workflow determines that the user device was located at the user's home 10 minutes ago, a route between the user's home and the user's office takes 20 minutes to traverse, and the user device is currently located near a midpoint of that route, the unknown location may be characterized as being 'in transit to work' or something similar. Likewise, an unknown location may be characterized in accordance with

various environmental attributes detected by the user device. For example, if the user device is at an unknown public location with no Wi-Fi networks that the user device is configured to access but the user device can access a cellular network, the unknown location may be consistent with a known characterization of a device being at a 'public location without secure Wi-Fi access,' such that the user device can be characterized as being at a 'public location without secure Wi-Fi access.'

In these examples, readers will appreciate that a library or catalog of known characterizations may be created by observing many devices over time. In fact, each entry (i.e., each known characterization of a location) in the catalog may associated with location information that tends to be associated with a particular characterization. For example, devices may be monitored to determine that when a device is placed in a silent mode between the hours of 8 AM-1 PM on a Sunday, and a touch screen display for the device is not being interacted with by the user, and a microphone in the device detects periods of singing, the device is at a location that is characterized as a 'religious services' location. As such, information that was gathered **902** and is associated with an unknown location may be compared to signatures for the entries in the catalog of characterizations to determine whether the unknown location should be characterized using one of the entries in the catalog.

Readers will appreciate that the characterization of an unknown location may be associated with a set of expected device behaviors. For example, when a user device is characterized as being 'in transit to work', it may be expected that utilization of the user device may be extremely limited as the user may be operating a vehicle. Likewise, when a user device is characterized as being at a 'public location without secure Wi-Fi access', it may be expected that the user device will not be used to access sensitive financial data or initiate substantial monetary transactions on behalf of the user's employer.

The example method depicted in FIG. **9** also includes determining **916**, based on the characterization of the unknown location, whether device utilization is anomalous.

Determining **916** whether device utilization is anomalous may be carried out described above, generally by determining the extent to which device activity is consistent with normal activity that would be expected to occur when the user device is at an unknown location that is characterized in a particular way. Readers will appreciate that a determination so to whether device utilization is anomalous may only be based in part on the characterization of the location, as other information may also be taken into consideration (e.g., information from other devices, specific details around how the device is being used, policies that restrict how a device should be utilized at certain locations, etc. . . .).

For further explanation, FIG. **10** sets forth a flow chart illustrating an additional example method of establishing a location profile for a user device in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **10** is similar to the example method depicted in FIG. **9** , as the example depicted in FIG. **10** also includes gathering **902** information associated with the location of a user device, determining **904** whether the user device is being accessed at a known location, determining **908** a characterization of a location, and determining **910** whether device utilization is anomalous.

In the example method depicted in FIG. **10** , determining **904** whether the user device is being accessed at a known location can include determining **1002** whether the information associated with the location of a user device matches, at least within a predetermined threshold, location information associated with one or more location profiles. A location profile may be embodied, for example, as a data structure that associates one or more known locations with information

associated with the known locations. The information that is associated with the known locations can include, for example, one or more device behaviors that are typically encountered when a device is at the known location, a geographic boundary that defines the known location, one or more characterizations of the known location, temporal aspects of device and user behavior that are typically observed at the known location, and so on. As such, the location profile may be used to correlate such information with a known location, so that the location profile can be used to determine whether a particular device is at the known location.

In the example method depicted in FIG. **10** , the one or more location profiles may be created based on activity associated with the user device. For example, a location profile that is associated with the user device being at a hotel, on an airplane, at a coffee shop, at home, at work, or at some other location may be based on monitored or observed device activity or user activity associated with the user device itself. Alternatively, the one or more location profiles may be created based on activity associated with other devices. Such other devices may be embodied as, for example, similar devices for users that are part of the same organization (e.g., the same company, the same department within a company), devices for users that are determined to perform similar roles (e.g., engineers, accountants, legal team members) within an organization, and so on. In such a way, a particular user device may be monitored with the benefit of knowledge gleaned from monitoring other devices, which may be particularly useful when a particular user device is new and does not have a long history of monitored behavior.

In some embodiments, determining **910**, **914** whether device utilization is anomalous may be further based on a temporal profile of device activity. The temporal profile of device activity may include, for example, information describing days and times that particular applications are used on the user device, information describing days and times that the user device is typically being accessed or not being accessed, information describing the relative timing of different user or device activities (e.g., updated code is committed from the user's device to a code repository after text editing software or some other software used to write code has been utilized), or any other information that associates user behavior or device behavior with times at which such behavior is common, permitted, prohibited, and so on. The temporal profile may be embodied as a database, table, or some other data structure. In such embodiments, the relationship between times and device or user activities may be identified through the usage of one or more machine learning models that take activities (and the times associated with the activities) as inputs to the machine learning models.

For further explanation, FIG. **11** sets forth a flow chart illustrating an additional example method of establishing a location profile for a user device in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **11** is similar to the example methods depicted in FIG. **9** and FIG. **10** , as the example depicted in FIG. **11** also includes gathering **902** information associated with the location of a user device, determining **904** whether the user device is being accessed at a known location, determining **908** a characterization of a location, and determining **910** whether device utilization is anomalous based on the characterization of the known location. The example method depicted in FIG. **11** also includes creating **1102** a polygraph associated with the user device. Creating **1102** a polygraph associated with the user device may be carried out as described elsewhere in the present disclosure. Examples of such device-specific polygraphs are included herein.

In the example depicted in FIG. **11** , determining **904** whether the user device is being accessed at a known location can include determining **1104**, based on one or more devices that are physically proximate to the user device, a location of the user device. Readers will appreciate

that the exact distance between the user device and the one or more other devices (e.g., one or more routers that emit a Wi-Fi signal) may not be needed in order to determine **1104** a location of the user device. The term 'physically proximate' that is used here can be a threshold value that may even be a function of the particular wireless communications protocol that is being used. For example, if the user device can detect the presence of another device using a Bluetooth adapter, the user device may be assumed to be within 20 feet of the other device. If the user device can detect the presence of another device using a Wi-Fi adapter, however, the user device may be assumed to be within 100 feet of the other device. Readers will appreciate that these distance and technologies are included only as examples.

Consider an example in which a user device is embodied as a laptop computer and the laptop computer can detect the presence of five distinct networks using its Wi-Fi adapter. In such an example, upon detecting the presence of each network, resources may be accessed that associate the access ID of the network with a physical location associated with the network. For example, Google maintains a database that can be accessed via various APIs to determine the latitude and longitude associated with a wireless network's access ID. In some examples, this database may be queried with the access ID of the detected network to receive latitude and longitude information associated with the network. Readers will appreciate that, over time, the results obtained from querying such a database may be cached such and reused. For example, if location information for each of the five distinct networks is obtained and the same user turns on a smartphone that detects the same five networks, the cached results of the previously executed queries may be utilized to determine the first geolocation of the smartphone. Cached results may similarly be used for other users whose devices detect the same networks.

In the example depicted in FIG. **11** , determining **904** whether the user device is being accessed at a known location can include determining **1106**, based on data communications involving the user device, a location of the user device. Determining **1106** a location of the user device based on data communications involving the user device may be carried out, for example, by examining data communications traffic sent by the device and extracting the client IP address that is associated with the data communications traffic. The client IP address may be used, for example, to query one or more of a variety of services that convert IP addresses to location information (e.g., a city/state, latitude and longitude, and so on). In an alternative embodiment, the IP address of the user device may be identified in other ways other than inspecting network traffic (e.g., through the use of CLI commands such as ipconfig).

Readers will appreciate that the location of a user device may be misidentified, for example, if the user device is connected to a virtual private network or if the user device is being operated in some other way where its IP address (or other data communications attribute) would not be an accurate representation of the device's location. As such, in some embodiments multiple pieces of information that are associated with a location can be used rather than relying exclusively on a single piece of information when determining a user device's location.

In some embodiments, a location cache may be maintained. As described above, as part of the process of determining a location of the user device, external resources may be accessed. For example, resources such as those maintained by Google may be accessed that associate the access ID of a network with a physical location associated with the network. Such resources, however, may not be free to access, may be timely to access, the resource may return location information in a less than ideal format (e.g., the resource return latitude and longitude information when the desired format of the location is a zip code), or may be undesirable to

access for other reasons. As such, the results obtained from querying such resources may be cached and reused in a local location cache.

Such a location cache may be a data repository such as a database, a file, a table, or embodied in some other way. In some embodiments, the location cache may be 'local' in the sense that it is stored on the user device itself or stored in some location that does not suffer from the same undesirable characteristics as the original resource. Continuing with the example where a Google database may be accessed (for a monetary charge) that associate the access ID of a network with a physical location associated with the network, after that initial access the location information associated with the access ID of the network may be maintained in a database that the user device can access free of charge. Readers will appreciate that in these examples, the content contained in the local location cache may be stored in the format desired by the user device (e.g., stored as city/state information rather than GPS coordinates).

Readers will appreciate that some embodiments may include a different combination of the steps described above, including variations of such steps. For example, some embodiments of establishing a location profile for a user device may include gathering information associated with a location of a user device, determining a characterization of the location, and determining, based on a characterization of the known location, whether device utilization is anomalous such that the step of determining whether a location is known is optional.

For further explanation, FIG. **12** sets forth a flow chart illustrating an example method of detecting deviation from normal behavior of a user device in accordance with some embodiments of the present disclosure.

The example method depicted in FIG. **12** includes generating **1202**, using information describing historical activity associated with a user device, a trained model for detecting normal activity for the user device. The trained model may be embodied, for example, as a model artifact that is created by a training process in which machine learning algorithms are provided with training data to learn from. The trained model, once deployed, can make predictions, identify patterns, and perform other functions on new data (i.e., data that was not part of the training data).

Generating **1202** a trained model for detecting normal activity for the user device using information describing historical activity associated with a user device may therefore be carried out, for example, by applying one or more machine learning algorithms to a training dataset that includes the information describing the historical activity associated with the user device. The information describing the historical activity associated with the user device can include, for example, information describing the locations at which the user device was utilized at some point in the past, information describing the applications on the user device that were executed at some point in the past, information describing the dates and times the applications on the user device that were executed, and so on. In such an example, activity associated with the user device may be deemed to be 'historical' if the activity occurred at some point before training occurred.

Consider an example in which training occurred at time t_0. In such an example, any activity associated with the user device that occurred prior to time to would be 'historical' activity associated with the user device. Further assume that additional training occurred at time t_1, which is later than time t_0. In this example, activity associated with the user device that occurred prior to time t_1 would be 'historical' activity associated with the user device, at least with respect to the additional training (whereas activity that occurred between time to and time t_1 would not be 'historical' activity with respect to the original training that occurred at time to).

The example method depicted in FIG. **12** also includes gathering **1204** information describing current activity associated with the user device. The information current activity associated with

the user device may be embodied, for example, as information describing activity associated with the user device that was gathered **1204** at a point in time such that it was not available for inclusion in the set of training data that was used to generate (or further refine via additional training) the trained model. The information describing current activity of the user device may include the types of information described above (e.g., location of the user device, applications executed by the user device, and much more). The information describing the current activity of the user device may be gathered **1204**, for example, by the agents described above including agents that are executing on the user device itself.

The example method depicted in FIG. **12** also includes determining **1206**, by using the information describing current activity associated with the user device as input to the trained model, whether the user device has deviated from normal activity. Determining **1206** whether the user device has deviated from normal activity by using the information describing current activity associated with the user device as input to the trained model may be carried out, for example, by executing the trained model on the user device and coupling the trained model to a data stream, data repository, or other source for the information describing current activity associated with the user device. In such an example, such information may describe 'current' activity in the sense that the information represents activity that has occurred in some recent period of time (e.g., the last minute, the last second), activity that is being monitored in real-time, activity that has occurred since the model was most recently trained, or activity that is 'current' as determined by some other rule or heuristic.

For further explanation, FIG. **13** sets forth a flow chart illustrating an example method of sets forth a flow chart illustrating an example method of detecting deviation from normal behavior of a user device in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **13** is similar to the example method depicted in FIG. **12** , as FIG. **13** also includes generating **1202** a trained model for detecting normal activity for the user device, gathering **1204** information describing current activity associated with the user device, and determining **1206** whether the user device has deviated from normal activity.

In the example method depicted in FIG. **13** , generating **1202** a trained model for detecting normal activity for the user device can include generating **1302** the trained model using information describing physical locations at which the user device was utilized. As described above, the physical locations at which the user device was utilized may be determined from a combination of sources (e.g., the location of physically proximate devices, data communications characteristics of the user device, user input, location-detecting devices that are embedded within the user device). Such locations at which the user device was utilized may be included in a training dataset that is used to generate the trained model. In fact, the trained model may ultimately identify density clusters such that locations that are not identical matches are determined to be the same location. For example, a cluster of locations at which the user device was utilized may be identified as locations within the user's office complex, such that the user being on the northeast side of the office building and the user being on the southwest side of the office building are not treated as distinct locations, but are instead treated as the user being within the same density area that represents a single logical location. Likewise, a cluster of locations that represent a commonly traversed path may be treated as a single entity. For example, the trained model may learn a user's daily route to work and may treat all of the individual locations along that route as a single entity. In some embodiments, particular locations may be labelled (e.g., home, office, etc. . . .) as part of the training process or after the model has been trained.

In the example method depicted in FIG. **13** , generating **1202** a trained model for detecting normal activity for the user device can include generating **1304** the trained model using information describing usage of one or more applications executed on the user device. The information describing usage of one or more applications executed on the user device may include, for example, information describing when the applications were used, information describing what features of the applications were used, information describing what external data sources were accessed when the applications were being used, information describing what data communications occurred when the applications were being used, and so on. Such information may be included in a training dataset that is used to generate the trained model. In fact, training the model using the training datasets may reveal relationships between different applications, patterns related to how the user utilizes the applications, actions that trigger the usage of the application (e.g., the user typically accesses an email application after receiving a notification), and many other patterns that represent normal activity. As such, the trained model may be configured to identify deviations from normal activity.

In the example method depicted in FIG. **13** , generating **1202** a trained model for detecting normal activity for the user device can include generating **1306** the trained model using information describing times at which activity previously occurred. The information describing times at which activity previously occurred may be embodied, for example, as absolute values (e.g., at 12:36 PM on a specific date), as relative values (e.g., after using application X, within 30 minutes after turning on the user device), as a value that is expressed in its relationship to some other activity or detected condition (e.g., while the user was in transit to work, while the user was at work), or in some other way. Such information may be included in a training dataset that is used to generate the trained model. In fact, training the model using the training datasets may reveal temporal relationships between different activities, patterns related to an ordering of a set of activities or detected conditions, and many other patterns that represent normal activity. As such, the trained model may be configured to identify deviations from normal activity.

Readers will appreciate that while the embodiments described above are largely described as individual types of information (e.g., location information, temporal information, device usage information) that may be used to generate **1202** a trained model for detecting normal activity for the user device, readers will appreciate that the process of generating **1202** a trained model for detecting normal activity for the user device may actually include one or more machine learning algorithms ingesting combinations of these types of information. In fact, other types of information may also be used in the process of generating **1202** a trained model for detecting normal activity for the user device. Readers will appreciate that any information that can be captured by an agent that is executing on the user device, including combinations thereof, can be used when generating **1202** a trained model for detecting normal activity for the user device.

In the example method depicted in FIG. **13** , gathering **1204** information describing current activity associated with the user device can include gathering **1308** information describing a physical location at which the user device is currently being used. The information describing a physical location at which the user device is currently being used may be gathered **1308**, for example, by one or more agents that are executing on the user device, by one or more agents that are executing on other devices (e.g., on a device that communicates with the user device over a local network). Such information may include, for example, information that itself does not represent a location of the user device but which may be used to determine the location of the user device as described above. Such information may include information describing other devices or network that are detected by the user device, information describing data

communications characteristics of the device, and so on. In other embodiments, the information itself may represent a location of the user device. For example, such information may include global positioning system ('GPS') coordinates obtained from a GPS receiver in the user device.

In the example method depicted in FIG. **13** , gathering **1204** information describing current activity associated with the user device can include gathering **1310** information describing usage of applications that are being accessed by the user device. The information describing usage of applications that are being accessed by the user device may be gathered **1310**, for example, by one or more agents that are executing on the user device, by one or more agents that are executing on a device/service that executes the application, or in other ways described above. Such information may include, for example, information describing which applications are being executed, information describing what data communications networks are being accessed when executing an application, information describing particular features of the application that are being utilized, and many others.

In the example method depicted in FIG. **13** , gathering **1204** information describing current activity associated with the user device can include gathering **1312** information describing times at which current activity occurred on the device. The information describing times at which current activity occurred on the device may be gathered **1312**, for example, by one or more agents that are executing on the user device, by one or more agents that are executing elsewhere, by inspecting activity logs, or in some other way. Such information may include, for example, information describing the exact time that some activity occurred, a range of times during which the activity occurred, a relative time during which the activity occurred, and so on.

For further explanation, FIG. **14** sets forth a flow chart illustrating an example method of sets forth a flow chart illustrating an example method of detecting deviation from normal behavior of a user device in accordance with some embodiments of the present disclosure. The example method depicted in FIG. **14** is similar to the example method depicted in FIGS. **12** and **13** , as FIG. **14** also includes generating **1202** a trained model for detecting normal activity for the user device, gathering **1204** information describing current activity associated with the user device, and determining **1206** whether the user device has deviated from normal activity.

The example method depicted in FIG. **14** also includes periodically **1402** retraining the trained model. Periodically **1402** retraining the trained model may be carried out, for example, by including recently acquired data describing various aspects of the user device's operation and usage into a training data that is used to train the model. Such data may be 'recently acquired' in the sense that it was not included in training data that was previously used to train the model. The recently acquired data may be used either alone or in combination with training data that was previously used to train the model as part of a process to retraining the trained model, at which point the retrained model may be deployed on the user device. Readers will appreciate that retraining the trained model periodically such as, for example, according to a predetermined schedule, upon the satisfaction of some condition (e.g., a threshold amount of new data has been acquired, sufficient resources to carry out the retraining have become available, alerts are being generated at a threshold level indicating that perhaps the trained model does not sufficiently understand normal behavior), upon request from a user or administrator, or in some other way. Through such periodically **1402** retraining the trained model, the trained model may be an evolving entity that can differentiate between normal and abnormal activity even as a user's interactions with the device or usage of the device change over time.

The example method depicted in FIG. **14** also includes generating **1404** an alert after determining that the user device has deviated from normal activity. The alert may be

generated **1404** and expressed through one or more of the polygraphs described above. In other embodiments, alerts may be delivered in some other way (e.g., as a notification that is sent to some predetermined recipient using some predetermined delivery mechanism).

The example method depicted in FIG. **14** also includes initiating **1406** a remediation workflow after determining that the user device has deviated from normal activity. Initiating **1406** a remediation workflow may be carried out as part of an auto-remediation capability that may be provided to the monitored devices. The remediation workflow may be configured to perform a variety of tasks including, for example, restricting the access of the user device to certain data, restricting the usage of certain applications on the user device, enabling some feature on the user device (e.g., communications over an unsecured network is detected so a data encryption feature for data communications is enabled), or performing some other function. In some embodiments, the remediation workflow may be designed to either prevent the abnormal activity from occurring or even enabling the abnormal activity upon the satisfaction of some condition (e.g., an administrator approves the activity, the user authenticates their identity).

For further explanation, FIG. **15**A sets forth an example of a user-specific polygraph **1500** in accordance with some embodiments of the present disclosure. The user-specific polygraph **1500** depicted in FIG. **15**A may be generated in a similar manner as the polygraphs described above and may have similar features and capabilities. The user-specific polygraph **1500** depicted in FIG. **15**A includes a representation of a user and information describing the user, in this case designated as User ABC **1502**. The representation of the user may include, for example, a user's name, a user's handle, or any other information associated with the user. Such information associated with the user may be visible in the original presentation of the polygraph or may be accessible in other ways (e.g., hovering a mouse over the user icon).

The user-specific polygraph **1500** depicted in FIG. **15**A also includes information describing the location of the user (as represented by the location of the device that is being used by the user), depicted here as San Francisco, CA **1504**. The location may be obtained as described in greater detail above. The representation of the location of the user may include, for example, an identification of a city, geographical coordinates, an identification of a known location (e.g., home, office), or any other location information associated with the user. Such location information associated with the user may be visible in the original presentation of the polygraph or may be accessible in other ways (e.g., hovering a mouse over the user icon).

The user-specific polygraph **1500** depicted in FIG. **15**A also includes information describing the user device, depicted here as user device MRD1 **1506**. The information describing the user device may include, for example, a device name, a label for the device (e.g., phone, laptop, work laptop) or any other information describing the device. Such device information may be visible in the original presentation of the polygraph or may be accessible in other ways (e.g., hovering a mouse over the user icon).

The user-specific polygraph **1500** depicted in FIG. **15**A also includes information describing applications accessed by the device, depicted here as internal sales application **1508**, web browser application **1510**, ad messaging application **1512**. Information describing the applications that are accessed by the user device may be obtained as described in greater detail above. The representation of the applications may include, for example, a name of the application, a name of the binary, a custom label for the application, or any other information associated with the application including information describing the usage of the application.

Such application-related information may be visible in the original presentation of the polygraph or may be accessible in other ways (e.g., hovering a mouse over the user icon).

The user-specific polygraph **1500** depicted in FIG. **15**A also includes information describing specific details regarding how a particular application is being used. For example, the internal sales application **1508** is depicted as connecting to an internal sales database **1514**. Information associated with the internal sales database (e.g., where the database is hosted, what credentials were used to access the database, how many queries have been directed to the database) may also be depicted or may be accessible in other ways (e.g., hovering a mouse over the user icon).

Likewise, the web browser application **1510** is depicted as being connected to different endpoints, which may be carried out through the usage of different tabs or different instances of the web browser application **1510**. For example, the web browser application **1510** is depicted as accessing a social media **1514** site, a productivity **1518** site (e.g., Salesforce, a web-based repository), a bandwidth intensive **1520** site such as a streaming video site, and a site that requires private information **1522** such as a banking site. In some embodiments, additional information associated with each endpoint **1516**, **1518**, **1520**, **1522** may be displayed or may be accessible in other ways (e.g., hovering a mouse over the user icon). In fact, the links between different icons in polygraph may also be enriched with data. For example, the links between the web browser application **1510** and the endpoints **1516**, **1518**, **1520**, **1522** may include information describing how long the connection has been established, how much data has been transferred since the connection was established, and so on. In other embodiments, other data associated with the links between two icons may also be enriched with data that may be visible in the default view of the polygraph or accessible in some other way as described above.

For further explanation, FIG. **15**B sets forth an example of a user-specific polygraph **1500** in accordance with some embodiments of the present disclosure. The example depicted in FIG. **15**B illustrates an embodiment in which an alert **1526** is presented in the polygraph **1500**. In this particular example, the alert **1526** is generated for a messaging application **1512** that is connected to an unknown network **1524**. As illustrated in the alert **1526**, a malicious IP address has been identified in the unknown network **1524**. Although not illustrated in this example, the alert **1526** may be coupled with functionality that is accessible in the displayed **1526** alert, which such functionality can include ignoring the alert, terminating the application, initiating a remediation workflow, or taking some other action.

Readers will appreciate that the polygraph **1500** depicted in FIGS. **15**A and **15**B is just one example of a user-specific polygraph that may be generated and utilized as described above. In other embodiments, less or additional information may be included in the polygraph **1500**, different information may be included in the polygraph **1500**, different actions may be initiated via the polygraph **1500**, or the polygraph **1500** may otherwise differ from the depicted examples.

For further explanation, FIG. **16** sets forth an example method for using activity monitored by multiple data sources to identify shadow systems according to some embodiments of the present disclosure. The method of FIG. **16** may be performed, for example, in a distributed edge platform as described above, or another computing environment or execution environment as can be appreciated.

The method of FIG. **16** includes gathering **1602** first information describing access to one or more resources by one or more user devices of a user. The one or more user devices are associated with the user via a variety of approaches. For example, the one or more user devices may include one or more known user devices registered or having a profile that associates the user device with the user. As another example, the one or more user devices may be logged into

a distributed edge platform or other service provider using authentication credentials for the user. As a further example, the one or more user devices may have accessed an identity provider using authentication credentials for the user. The one or more user devices may be associated with a particular user by other approaches as can be appreciated.

The one or more resources accessed by the one or more user devices are remotely disposed resources accessible via a network. For example, the one or more resources may include one or more applications (e.g., software as a service (SaaS) applications from third party service providers deployed or managed by an organization including or overseeing the user), one or more private applications (e.g., applications deployed in a data center or other private computing environment controlled by an organization including or overseeing the user), one or more documents, one or more websites, and potentially other network-accessible resources as can be appreciated.

The first information is gathered **1602** from one or more data sources other than the one or more user devices. In some embodiments, the first information is gathered **1602** from the one or more resources or one or more providers of the one or more resources. In some embodiments, the first information may include log entries describing access to the particular resources. For example, the first information may include logs generated by the applications themselves or application service providers, by hosting services for particular webpages, by storage systems storing access documents or files, and the like. The first information may be gathered **1602** via exposed application program interfaces (APIs), by accessing one or more accessible files encoding the first information, and the like.

The first information may describe previous or historical access to the resources by particular users, by particular user devices, and the like. For example, the first information may describe an access to a resource and indicate particular authentication credentials, usernames, accounts, and the like used to access that resource. As another example, the first information may describe access to a resource and indicate device attributes for a device accessing the resource, including an internet protocol (IP) address, network interface address (e.g., a MAC address), a device identifier, and the like. As a further example, the first information may describe attributes of a network request or network address used to access a resource, including source and destination IP addresses, source and destination ports, particular protocols used to facilitate the request, descriptions of encryption protocols used to facilitate the request, certificates or keys used to facilitate the request, and the like. The first information may also include identifiers for particular accessed resources, including particular applications being accessed, particular websites, documents, or files being accessed, and the like. The first information may also indicate a particular type of transaction or access being performed via the request (e.g., a read access or a write access).

In some embodiments, the first information is gathered **1602** from an identity provider used to access the one or more resources. An identity provider allows access to multiple resources using a shared set of authentication credentials. For example, a user of a particular organization may log into the identity provider in order to access a variety of applications deployed or managed by the organization, rather than require the user to individually log into each of multiple applications. Accordingly, the identity provider may maintain logs describing particular Accordingly, in some embodiments, the first information may describe uses of the identity provider by a particular user (e.g., using a particular set of authentication credentials). For example, the first information may include various attributes identifying a particular user device used to access the identity provider, including an IP address, network interface address, location

information, or other information describing a device used to access the identity provider. The first information may also include times at which access to the identity provider was requested. The first information may also describe failed attempts to access the identity provider, including incorrect authentication credentials, device identifiers, and the like. Thus, portions of the first information describing a failed access to the identity provider may be associated with a particular user by virtue of using a user device known to be associated with the user.

In some embodiments, the first information from the identity provider may describe access to particular resources (e.g., SaaS applications or other resources) via the identity provider. Thus, access to a resource may be described using data from an identity provider, data from the resource itself, or both. Where two-factor authentication or other supplemental authentication schemes are used, the first information may describe a method of two-factor authentication used, values or data points used in the two-factor authentication, and so forth.

The first information may be requested on a per device basis using particular device attributes, on a per user basis using a username for the user, and combinations thereof. For example, an API request or other request may be provided to a particular data source (e.g., an identity provider, a SaaS application) to retrieve first information records having a particular username, a particular device identifier, or both. Such a request may also include other delimiters or filters, including particular time ranges, particular ranges of IP addresses, and so forth.

One skilled in the art will appreciate that the particular data included in the first information is dependent on what information is visible to the particular data source providing the first information. For example, a SaaS service provider may have access to particular data describing access to the SaaS service provider (e.g., logs of the SaaS application) that may differ from information accessible or visible to an identity provider.

The method of FIG. **16** also includes gathering **1604**, from at least a subset of the one or more user devices, second information describing access to the one or more resources. For example, assume that a user device is configurable to provide information (e.g., the second information) to a distributed edge platform or other computing environment using a client application executed on the user device. Such a client application may include, for example, a proxy application or an application implementing proxy functionality such that network traffic from the user device, including requests to access the one or more resources, is routed to the distributed edge platform or another destination.

Such a client application may also provide, to the distributed edge platform, metadata describing network requests or device attributes. In some embodiments, the metadata is generated by the client application and included with or appended to network requests routed to a proxy destination. In some embodiments, the metadata is generated by the client application and sent in a separate data payload or data stream. The second information may include a variety of data points or attributes as described above with respect to describing behavior of a user or a user device, including device identifiers, user identifiers, location identifiers, device attributes (e.g., an operating system or operating system version, a hardware configuration), device performance attributes (e.g., memory usage, processor usage, bandwidth usage, and the like), accessed resource identifiers, times at which particular network requests or resource accesses occurred, and the like. The second information may also describe network requests for accessing or logging into an identity provider, such as an identifier for the identity provider, a time at which the access occurred, authentication credentials used to access the identity provider, and the like. The second information is gathered from at least a subset of the one or more user devices due to the second information being gathered via a client application. Where a user device does not use

the client application, or use of the client application is circumvented somehow, second information may not be gathered from that user device. Although the present discussion describes gathering **1604** second information from at least a subset of the user devices, one skilled in the art will appreciate that, in some embodiments, no portions of second information may be gathered from any user devices. Accordingly, a lack of gathering any second information from any user device is also contemplated within the scope of the present disclosure. Thus, a null set of second information is also contemplated in the scope of the present disclosure.

The method of FIG. **16** also includes identifying **1606** one or more shadow systems based on a discrepancy between the first information and the second information. As described herein, "shadow systems" are user devices engaging in "shadow information technology (IT)" as would be appreciated by one skilled in the art, whereby access to a resource is performed outside of the operational control of an IT department. For example, in some embodiments, a user device associated with a user of an organization may be used to access applications (e.g., SaaS applications, web-based applications) deployed or managed outside of the organization (e.g., "shadow applications"). As another example, a user device associated with the user may be used to access applications deployed or managed by the organization, but the user device is not accessing the application by a designated methodology. For example, the user device may be accessing the application without the use of a client application as described above that performs proxy functionality for redirecting network requests to a distributed edge platform or other execution environment. As a further example, the user device may be accessing the application using an unauthorized, unknown, or malicious virtual private network (VPN) or proxy service. Such user devices are hereinafter referred to as "shadow systems."

As is set forth above, both the first information and the second information describe access to the one or more resources from different perspectives. The first information is generated from the perspective of an identity provider or SaaS accessed via a user device, and the second information is generated from the perspective of a user device via a client application executed on the user device. Each access to a particular resource by a user device should be described in first information gathered from the identity provider, from the resource itself, or both. In contrast, access to a resource by a user device will only be described in second information from that user device if the access was performed via the client application (e.g., via a proxy application or by proxy functionality of that application). Thus, where a user device accesses a resource without using the client application or otherwise circumventing the client application, that access will be described in the first information but not described in any second information from the user device.

Accordingly, in some embodiments, identifying **1608** the shadow systems includes identifying **1608** accesses described in the first information and not described in the second information and identifying **1610**, as one or more shadow systems, one or more user devices corresponding to the one or more accesses. A user device that performed the identified **1608** accesses is thus identified as a "shadow system." In other words, identifying **1606** the shadow systems includes identifying one or more shadow systems by identifying **1608** accesses described in the first information and not described in the second information and identifying, as a shadow system, one or more devices that performed the identified **1608** accesses.

In response to identifying **1606** the shadow systems, various remedial actions may be performed as is described in further detail below. One skilled in the art will appreciate that identifying **1606** one or more shadow systems improves security by identifying those user

devices circumventing organizational policies or security monitoring systems in order to access organizational resources.

A designation of a user device as a shadow system may affect detections of abnormal or risky user behavior according to the approaches described above. For example, a designation of a user device as a shadow system may factor in a composite score used to evaluate whether some use of the user device (e.g., some network activity) deviates from normal behavior of the user. As another example, designation of a user device as a shadow system may be provided as input to a model that identifies abnormal behavior for a user.

For further explanation, FIG. **17** sets forth a flowchart of another example method of using activity from multiple data sources to identify shadow systems according to some embodiments of the present disclosure. The method of FIG. **17** is similar to FIG. **16** in that the method of FIG. **17** includes gathering **1602** first information describing access to one or more resources by one or more user devices of a user; gathering **1604**, from at least a subset of the one or more user devices, second information describing access to the one or more resources; and identifying **1606** one or more shadow systems based on a discrepancy between the first information and the second information by identifying **1608** accesses described in the first information and not described in the second information and identifying **1610**, as one or more shadow systems, one or more user devices corresponding to the one or more accesses.

The method of FIG. **17** differs from FIG. **16** in that the method of FIG. **17** also includes generating **1702** a polygraph indicating the one or more shadow systems. Such a polygraph may be generated according to any approach for generating polygraphs as described above.

An example polygraph **2000** for indicating shadow systems is set forth in FIG. **20** .

The polygraph **2000** of FIG. **20** describes network activity with respect to a user, represented in the polygraph **2000** as User ABC **2002**.

The polygraph **2000** also includes a representation of a location associated with the user, shown as San Francisco, CA **2004**. The location of the user may be determined using similar approaches as are set forth above. The polygraph **2000** also includes representations of user devices associated with the user, shown as primary device **2006** and other device **2008**.

The polygraph **2000** also includes representations for executable applications (e.g., executables on the primary device **2006** or other device **2008**) used to access particular resources, shown as web browser application **2010** and web browser application **2012**. The polygraph **2000** also includes representations of SaaS applications accessed by the user via the user devices, shown as application **2014** and application **2016**.

Here, the one or more shadow systems are indicated in the polygraph using an alert **2018** included in the polygraph **2000**. The alert **2018** is anchored to the representation of the other device **2008**, indicating that the other device **2008** has been identified as a shadow system. As the polygraph **2000** only shows the other device **2008** accessing the application **2016**, it is understood that, in this example, the other device **2008** is identified as a shadow system by accessing the application **2016** in an unauthorized manor, such as without using a client application (e.g., a proxy application) that would generate second information describing access to the application **2016** by the other device **2008**.

Although the polygraph **2000** indicates shadow systems by anchoring an alert **2018** to an identified shadow system, one skilled in the art will appreciate that other approaches for indicating shadow systems may also be used. For example, an alert **2018** may be anchored to another node or to a connection between nodes. As another example, the representation of the other device **2008** may include a label, highlighting, flag, icon, or other distinguishing mark

signifying that the other device **2008** has been identified as a shadow system. As another example, one or more connections between nodes (e.g., between the other device **2008**, web browser application **2012**, or application **2016** may be highlighted or otherwise designated, thereby indicating that some access was performed by a shadow system.

For further explanation, FIG. **18** sets forth a flowchart of another example method of using activity from multiple data sources to identify shadow systems according to some embodiments of the present disclosure. The method of FIG. **18** is similar to FIG. **16** in that the method of FIG. **18** includes gathering **1602** first information describing access to one or more resources by one or more user devices of a user; gathering **1604**, from at least a subset of the one or more user devices, second information describing access to the one or more resources; and identifying **1606** one or more shadow systems based on a discrepancy between the first information and the second information by identifying **1608** accesses described in the first information and not described in the second information and identifying **1610**, as one or more shadow systems, one or more user devices corresponding to the one or more accesses.

The method of FIG. **18** differs from FIG. **16** in that the method of FIG. **18** also includes generating **1802** an alert in response to identifying the one or more shadow systems. The alert may be embodied in a variety of approaches. For example, the alert may be presented as an alert **2018** in a polygraph **2000** as set forth above. As another example, the alert may be embodied as a notification, text message, email, chat message, or other message sent to one or more recipients. Such recipients may include, the user, the shadow system, a manager, administrator, or other supervisor of the user, a security team member, or other recipients as can be appreciated.

In some embodiments, the alert may identify the shadow system. As another example, the alert may describe one or more accesses that caused the device to be identified as a shadow system. As another example, the alert may include links to polygraphs or other visualizations describing the shadow systems. As will be described in further detail below, the alert may be used to direct the user to a remediation workflow by including one or more links or other user interface elements that, when selected, cause a user device (e.g., a shadow system) to access a remediation workflow.

For further explanation, FIG. **19** sets forth a flowchart of another example method of using activity from multiple data sources to identify shadow systems according to some embodiments of the present disclosure. The method of FIG. **19** is similar to FIG. **16** in that the method of FIG. **19** includes gathering **1602** first information describing access to one or more resources by one or more user devices of a user; gathering **1604**, from at least a subset of the one or more user devices, second information describing access to the one or more resources; and identifying **1606** one or more shadow systems based on a discrepancy between the first information and the second information by identifying **1608** accesses described in the first information and not described in the second information and identifying **1610**, as one or more shadow systems, one or more user devices corresponding to the one or more accesses.

The method of FIG. **19** differs from FIG. **16** in that the method of FIG. **19** also includes directing **1902** the user to a remediation workflow in response to identifying the one or more shadow systems. A remediation workflow is one or more steps performed by a user device in order to remediate past shadow activity (e.g., activity that caused a user device to be identified as a shadow system), such as by preventing subsequent shadow activity or mitigating any potential harm that may have resulted from the shadow activity. For example, in some embodiments, a

user may be directed **1902** to a remediation workflow by sending an alert, link, message, and the like to an identified shadow system or other user device associated with the user.

In some embodiments, the remediation workflow may facilitate installation or repair of an existing installation of a client application used to access remote resources, such as a client application that performs proxy functions on the installed user device. Accordingly, in some embodiments, the remediation workflow may include links to download the client application for installation, descriptions for installation of the client application, and the like. In some embodiments, the remediation workflow may facilitate identification and remediation of potential harm from the shadow activity. For example, the remediation workflow may include links to malware detection or remediation software, instructions for installation and use of such software, and the like. The remediation workflow may also include steps for confirming completion of prior steps in the remediation workflow. For example, the remediation workflow may include a link that when selected, causes the user device to send a network request. The network request may then be analyzed by a recipient to determine if the network request was received via a particular proxy, indicating that the user successfully installed or repaired an instance of the client application. As another example, the remediation workflow may include a link or user interface element that causes malware detection or remediation software to execute, thereby ensuring that the software was run on the user device. One skilled in the art will appreciate that other remediation approaches may also be facilitated by a remediation workflow to which the user is directed **1902**.

One or more embodiments may be described herein with the aid of method steps illustrating the performance of specified functions and relationships thereof. The boundaries and sequence of these functional building blocks and method steps have been arbitrarily defined herein for convenience of description. Alternate boundaries and sequences can be defined so long as the specified functions and relationships are appropriately performed. Any such alternate boundaries or sequences are thus within the scope and spirit of the claims. Further, the boundaries of these functional building blocks have been arbitrarily defined for convenience of description. Alternate boundaries could be defined as long as the certain significant functions are appropriately performed. Similarly, flow diagram blocks may also have been arbitrarily defined herein to illustrate certain significant functionality.

To the extent used, the flow diagram block boundaries and sequence could have been defined otherwise and still perform the certain significant functionality. Such alternate definitions of both functional building blocks and flow diagram blocks and sequences are thus within the scope and spirit of the claims. One of average skill in the art will also recognize that the functional building blocks, and other illustrative blocks, modules and components herein, can be implemented as illustrated or by discrete components, application specific integrated circuits, processors executing appropriate software and the like or any combination thereof.

While particular combinations of various functions and features of the one or more embodiments are expressly described herein, other combinations of these features and functions are likewise possible. The present disclosure is not limited by the particular examples disclosed herein and expressly incorporates these other combinations.