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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
08 March 2018 (08.03.2018)

WIPO | PCT

(10) International Publication Number
WO 2018/044604 A1

(51) International Patent Classification:
G06F 21/41 (2013.01) *G06F 21/60* (2013.01)

(21) International Application Number:
PCT/US2017/047726

(22) International Filing Date:
21 August 2017 (21.08.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/381,866 31 August 2016 (31.08.2016) US
15/680,362 18 August 2017 (18.08.2017) US

(71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, Redwood Shores, California 94065 (US).

(72) Inventors: **WILSON, Gregg**; 3917 Walnut Clay Drive, Austin, Texas 78731 (US). **MEDAM, Venkateswara Reddy**; 4205 Noah Court, Modesto, California 95356 (US).

(74) Agent: **GOLDSMITH, Barry S.**; Miles & Stockbridge P.C., 1751 Pinnacle Drive, Suite 1500, Tysons Corner, Virginia 22102 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV,

(54) Title: DATA MANAGEMENT FOR A MULTI-TENANT IDENTITY CLOUD SERVICE

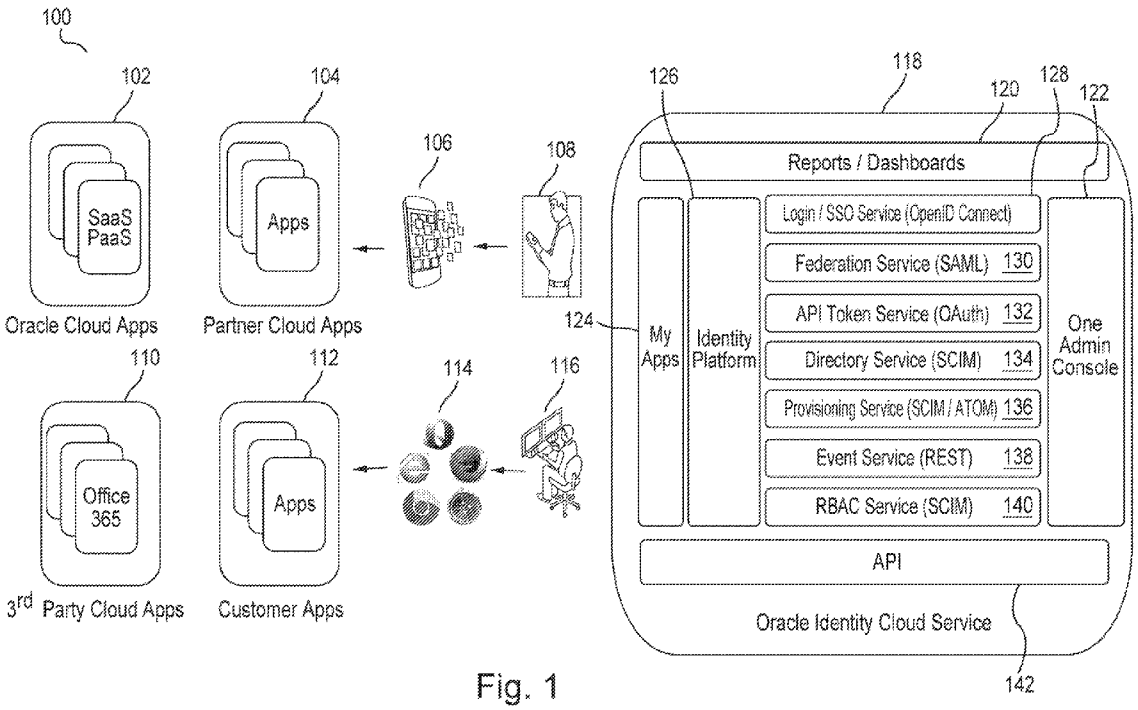

Fig. 1

(57) Abstract: Cloud based identity management is provided by receiving a request from an application by a web gate for a resource, where the request includes an operation on a resource type out of a plurality of resource types and the request specifies a tenant out of a plurality of tenants. Embodiments access a microservice based on the request, resolve the resource type, and validate that the operation is supported by the resource type based on metadata. Embodiments get a data provider associated with the tenant, call the data provider to perform the operation, and then return the resource.

[Continued on next page]

WO 2018/044604 A1

MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

DATA MANAGEMENT FOR A MULTI-TENANT IDENTITY CLOUD SERVICE

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority of U.S. Provisional Patent Application Serial No. 62/381,866, filed on August 31, 2016, the disclosure of which is hereby incorporated by reference.

FIELD

[0002] One embodiment is directed generally to identity management, and in particular, to identity management in a cloud system.

BACKGROUND INFORMATION

[0003] Generally, the use of cloud-based applications (e.g., enterprise public cloud applications, third-party cloud applications, etc.) is soaring, with access coming from a variety of devices (e.g., desktop and mobile devices) and a variety of users (e.g., employees, partners, customers, etc.). The abundant diversity and accessibility of cloud-based applications has led identity management and access security to become a central concern. Typical security concerns in a cloud environment are unauthorized access, account hijacking, malicious insiders, etc. Accordingly, there is a need for secure access to cloud-based applications, or applications located anywhere, regardless of from what device type or by what user type the applications are accessed.

SUMMARY

[0004] Embodiments provide cloud based identity management by receiving a request from an application by a web gate for a resource, where the request includes an operation on a resource type out of a plurality of resource types and the request specifies a tenant out of a plurality of tenants. Embodiments access a microservice based on the request, resolve the resource type, and validate that the operation is supported by the resource type based on metadata. Embodiments get a data provider associated with the tenant, call the data provider to perform the operation, and then return the resource.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] Figs. 1-5 are block diagrams of example embodiments that provide cloud-based identity management.

[0006] Fig. 6 is a block diagram providing a system view of an embodiment.

[0007] Fig. 6A is a block diagram providing a functional view of an embodiment.

[0008] Fig. 7 is a block diagram of an embodiment that implements Cloud Gate.

[0009] Fig. 8 illustrates an example system that implements multiple tenancies in one embodiment.

[0010] Fig. 9 is a block diagram of a network view of an embodiment.

[0011] Fig. 10 is a block diagram of a system architecture view of single sign on ("SSO") functionality in one embodiment.

[0012] Fig. 11 is a message sequence flow of SSO functionality in one embodiment.

[0013] Fig. 12 illustrates an example of a distributed data grid in one embodiment.

[0014] Fig. 13 illustrates a data manager architecture for an Identity Cloud Service ("IDCS") or an Identity as a Service ("IDaaS") in accordance with one embodiment.

[0015] Fig. 14 illustrates the functional flow of embodiments of the invention implemented by the resource data manager of Fig. 13.

[0016] Fig. 15 illustrates automatic schema versioning in accordance with one embodiment.

DETAILED DESCRIPTION

[0017] Embodiments implement metadata that defines a resource type and associated schemas. A request for performing an operation on a resource in a multi-tenant system is resolved using the metadata to determine the data provider associated with the tenant that performs the operation.

[0018] Embodiments provide an identity cloud service that implements a microservices based architecture and provides multi-tenant identity and data security management and secure access to cloud-based applications. Embodiments support secure access for hybrid cloud deployments (i.e., cloud deployments which include a combination of a public cloud and a private cloud). Embodiments protect applications

and data both in the cloud and on-premise. Embodiments support multi-channel access via web, mobile, and application programming interfaces ("APIs"). Embodiments manage access for different users, such as customers, partners, and employees. Embodiments manage, control, and audit access across the cloud as well as on-premise. Embodiments integrate with new and existing applications and identities. Embodiments are horizontally scalable.

[0019] One embodiment is a system that implements a number of microservices in a stateless middle tier environment to provide cloud-based multi-tenant identity and access management services. In one embodiment, each requested identity management service is broken into real-time and near-real-time tasks. The real-time tasks are handled by a microservice in the middle tier, while the near-real-time tasks are offloaded to a message queue. Embodiments implement access tokens that are consumed by a routing tier and a middle tier to enforce a security model for accessing the microservices. Accordingly, embodiments provide a cloud-scale Identity and Access Management ("IAM") platform based on a multi-tenant, microservices architecture.

[0020] One embodiment provides an identity cloud service that enables organizations to rapidly develop fast, reliable, and secure services for their new business initiatives. In one embodiment, the identity cloud service provides a number of core services, each of which solving a unique challenge faced by many enterprises. In one embodiment, the identity cloud service supports administrators in, for example, initial on-boarding/importing of users, importing groups with user members,

creating/updating/disabling/enabling/deleting users, assigning/un-assigning users into/from groups, creating/updating/deleting groups, resetting passwords, managing policies, sending activation, etc. The identity cloud service also supports end users in, for example, modifying profiles, setting primary/recovery emails, verifying emails, unlocking their accounts, changing passwords, recovering passwords in case of forgotten password, etc.

Unified Security of Access

[0021] One embodiment protects applications and data in a cloud environment as well as in an on-premise environment. The embodiment secures access to any application from any device by anyone. The embodiment provides protection across both environments since inconsistencies in security between the two environments may result in higher risks. For example, such inconsistencies may cause a sales person to continue having access to their Customer Relationship Management ("CRM") account even after they have defected to the competition. Accordingly, embodiments extend the security controls provisioned in the on-premise environment into the cloud environment. For example, if a person leaves a company, embodiments ensure that their accounts are disabled both on-premise and in the cloud.

[0022] Generally, users may access applications and/or data through many different channels such as web browsers, desktops, mobile phones, tablets, smart watches, other wearables, etc. Accordingly, one embodiment provides secured access across all these channels. For example, a user may use their mobile phone to complete a transaction they started on their desktop.

[0023] One embodiment further manages access for various users such as customers, partners, employees, etc. Generally, applications and/or data may be accessed not just by employees but by customers or third parties. Although many known systems take security measures when onboarding employees, they generally do not take the same level of security measures when giving access to customers, third parties, partners, etc., resulting in the possibility of security breaches by parties that are not properly managed. However, embodiments ensure that sufficient security measures are provided for access of each type of user and not just employees.

Identity Cloud Service

[0024] Embodiments provide an Identity Cloud Service ("IDCS") that is a multi-tenant, cloud-scale, IAM platform. IDCS provides authentication, authorization, auditing, and federation. IDCS manages access to custom applications and services running on the public cloud, and on-premise systems. In an alternative or additional embodiment, IDCS may also manage access to public cloud services. For example, IDCS can be used to provide Single Sign On ("SSO") functionality across such variety of services/applications/systems.

[0025] Embodiments are based on a multi-tenant, microservices architecture for designing, building, and delivering cloud-scale software services. Multi-tenancy refers to having one physical implementation of a service securely supporting multiple customers buying that service. A service is a software functionality or a set of software functionalities (such as the retrieval of specified information or the execution of a set of operations) that can be reused by different clients for different purposes, together with

the policies that control its usage (e.g., based on the identity of the client requesting the service). In one embodiment, a service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.

[0026] In one embodiment, a microservice is an independently deployable service. In one embodiment, the term microservice contemplates a software architecture design pattern in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs. In one embodiment, microservices are small, highly decoupled services and each may focus on doing a small task. In one embodiment, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms (e.g., an HTTP resource API). In one embodiment, microservices are easier to replace relative to a monolithic service that performs all or many of the same functions. Moreover, each of the microservices may be updated without adversely affecting the other microservices. In contrast, updates to one portion of a monolithic service may undesirably or unintentionally negatively affect the other portions of the monolithic service. In one embodiment, microservices may be beneficially organized around their capabilities. In one embodiment, the startup time for each of a collection of microservices is much less than the startup time for a single application that collectively performs all the services of those microservices. In some embodiments, the startup

time for each of such microservices is about one second or less, while the startup time of such single application may be about a minute, several minutes, or longer.

[0027] In one embodiment, microservices architecture refers to a specialization (i.e., separation of tasks within a system) and implementation approach for service oriented architectures ("SOAs") to build flexible, independently deployable software systems. Services in a microservices architecture are processes that communicate with each other over a network in order to fulfill a goal. In one embodiment, these services use technology-agnostic protocols. In one embodiment, the services have a small granularity and use lightweight protocols. In one embodiment, the services are independently deployable. By distributing functionalities of a system into different small services, the cohesion of the system is enhanced and the coupling of the system is decreased. This makes it easier to change the system and add functions and qualities to the system at any time. It also allows the architecture of an individual service to emerge through continuous refactoring, and hence reduces the need for a big up-front design and allows for releasing software early and continuously.

[0028] In one embodiment, in the microservices architecture, an application is developed as a collection of services, and each service runs a respective process and uses a lightweight protocol to communicate (e.g., a unique API for each microservice). In the microservices architecture, decomposition of a software into individual services/capabilities can be performed at different levels of granularity depending on the service to be provided. A service is a runtime component/process. Each microservice is a self-contained module that can talk to other modules/microservices. Each

microservice has an unnamed universal port that can be contacted by others. In one embodiment, the unnamed universal port of a microservice is a standard communication channel that the microservice exposes by convention (e.g., as a conventional Hypertext Transfer Protocol ("HTTP") port) and that allows any other module/microservice within the same service to talk to it. A microservice or any other self-contained functional module can be generically referred to as a "service".

[0029] Embodiments provide multi-tenant identity management services. Embodiments are based on open standards to ensure ease of integration with various applications, delivering IAM capabilities through standards-based services.

[0030] Embodiments manage the lifecycle of user identities which entails the determination and enforcement of what an identity can access, who can be given such access, who can manage such access, etc. Embodiments run the identity management workload in the cloud and support security functionality for applications that are not necessarily in the cloud. The identity management services provided by the embodiments may be purchased from the cloud. For example, an enterprise may purchase such services from the cloud to manage their employees' access to their applications.

[0031] Embodiments provide system security, massive scalability, end user usability, and application interoperability. Embodiments address the growth of the cloud and the use of identity services by customers. The microservices based foundation addresses horizontal scalability requirements, while careful orchestration of the services addresses the functional requirements. Achieving both goals requires decomposition

(wherever possible) of the business logic to achieve statelessness with eventual consistency, while much of the operational logic not subject to real-time processing is shifted to near-real-time by offloading to a highly scalable asynchronous event management system with guaranteed delivery and processing. Embodiments are fully multi-tenant from the web tier to the data tier in order to realize cost efficiencies and ease of system administration.

[0032] Embodiments are based on industry standards (e.g., OpenID Connect, OAuth2, Security Assertion Markup Language 2 ("SAML2"), System for Cross-domain Identity Management ("SCIM"), Representational State Transfer ("REST"), etc.) for ease of integration with various applications. One embodiment provides a cloud-scale API platform and implements horizontally scalable microservices for elastic scalability. The embodiment leverages cloud principles and provides a multi-tenant architecture with per-tenant data separation. The embodiment further provides per-tenant customization via tenant self-service. The embodiment is available via APIs for on-demand integration with other identity services, and provides continuous feature release.

[0033] One embodiment provides interoperability and leverages investments in identity management ("IDM") functionality in the cloud and on-premise. The embodiment provides automated identity synchronization from on-premise Lightweight Directory Access Protocol ("LDAP") data to cloud data and vice versa. The embodiment provides a SCIM identity bus between the cloud and the enterprise, and allows for different options for hybrid cloud deployments (e.g., identity federation and/or synchronization, SSO agents, user provisioning connectors, etc.).

[0034] Accordingly, one embodiment is a system that implements a number of microservices in a stateless middle tier to provide cloud-based multi-tenant identity and access management services. In one embodiment, each requested identity management service is broken into real-time and near-real-time tasks. The real-time tasks are handled by a microservice in the middle tier, while the near-real-time tasks are offloaded to a message queue. Embodiments implement tokens that are consumed by a routing tier to enforce a security model for accessing the microservices. Accordingly, embodiments provide a cloud-scale IAM platform based on a multi-tenant, microservices architecture.

[0035] Generally, known systems provide siloed access to applications provided by different environments, e.g., enterprise cloud applications, partner cloud applications, third-party cloud applications, and customer applications. Such siloed access may require multiple passwords, different password policies, different account provisioning and de-provisioning schemes, disparate audit, etc. However, one embodiment implements IDCS to provide unified IAM functionality over such applications. Fig. 1 is a block diagram 100 of an example embodiment with IDCS 118, providing a unified identity platform 126 for onboarding users and applications. The embodiment provides seamless user experience across various applications such as enterprise cloud applications 102, partner cloud applications 104, third-party cloud applications 110, and customer applications 112. Applications 102, 104, 110, 112 may be accessed through different channels, for example, by a mobile phone user 108 via a mobile phone 106, by a desktop computer user 116 via a browser 114, etc. A web browser (commonly

referred to as a browser) is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. Examples of web browsers are Mozilla Firefox®, Google Chrome®, Microsoft Internet Explorer®, and Apple Safari®.

[0036] IDCS 118 provides a unified view 124 of a user's applications, a unified secure credential across devices and applications (via identity platform 126), and a unified way of administration (via an admin console 122). IDCS services may be obtained by calling IDCS APIs 142. Such services may include, for example, login/SSO services 128 (e.g., OpenID Connect), federation services 130 (e.g., SAML), token services 132 (e.g., OAuth), directory services 134 (e.g., SCIM), provisioning services 136 (e.g., SCIM or Any Transport over Multiprotocol ("AToM")), event services 138 (e.g., REST), and authorization services 140 (e.g., SCIM). IDCS 118 may further provide reports and dashboards 120 related to the offered services.

Integration Tools

[0037] Generally, it is common for large corporations to have an IAM system in place to secure access to their on-premise applications. Business practices are usually matured and standardized around an in-house IAM system such as "Oracle IAM Suite" from Oracle Corp. Even small to medium organizations usually have their business processes designed around managing user access through a simple directory solution such as Microsoft Active Directory ("AD"). To enable on-premise integration, embodiments provide tools that allow customers to integrate their applications with IDCS.

[0038] Fig. 2 is a block diagram 200 of an example embodiment with IDCS 202 in a cloud environment 208, providing integration with an AD 204 that is on-premise 206. The embodiment provides seamless user experience across all applications including on-premise and third-party applications, for example, on-premise applications 218 and various applications/services in cloud 208 such as cloud services 210, cloud applications 212, partner applications 214, and customer applications 216. Cloud applications 212 may include, for example, Human Capital Management ("HCM"), CRM, talent acquisition (e.g., Oracle Taleo cloud service from Oracle Corp.), Configure Price and Quote ("CPQ"), etc. Cloud services 210 may include, for example, Platform as a Service ("PaaS"), Java, database, business intelligence ("BI"), documents, etc.

[0039] Applications 210, 212, 214, 216, 218, may be accessed through different channels, for example, by a mobile phone user 220 via a mobile phone 222, by a desktop computer user 224 via a browser 226, etc. The embodiment provides automated identity synchronization from on-premise AD data to cloud data via a SCIM identity bus 234 between cloud 208 and the enterprise 206. The embodiment further provides a SAML bus 228 for federating authentication from cloud 208 to on-premise AD 204 (e.g., using passwords 232).

[0040] Generally, an identity bus is a service bus for identity related services. A service bus provides a platform for communicating messages from one system to another system. It is a controlled mechanism for exchanging information between trusted systems, for example, in a service oriented architecture ("SOA"). An identity bus is a logical bus built according to standard HTTP based mechanisms such as web

service, web server proxies, etc. The communication in an identity bus may be performed according to a respective protocol (e.g., SCIM, SAML, OpenID Connect, etc.). For example, a SAML bus is an HTTP based connection between two systems for communicating messages for SAML services. Similarly, a SCIM bus is used to communicate SCIM messages according to the SCIM protocol.

[0041] The embodiment of Fig. 2 implements an identity ("ID") bridge 230 that is a small binary (e.g., 1 MB in size) that can be downloaded and installed on-premise 206 alongside a customer's AD 204. ID Bridge 230 listens to users and groups (e.g., groups of users) from the organizational units ("OUs") chosen by the customer and synchronizes those users to cloud 208. In one embodiment, users' passwords 232 are not synchronized to cloud 208. Customers can manage application access for users by mapping IDCS users' groups to cloud applications managed in IDCS 208. Whenever the users' group membership is changed on-premise 206, their corresponding cloud application access changes automatically.

[0042] For example, an employee moving from engineering to sales can get near instantaneous access to the sales cloud and lose access to the developer cloud. When this change is reflected in on-premise AD 204, cloud application access change is accomplished in near-real-time. Similarly, access to cloud applications managed in IDCS 208 is revoked for users leaving the company. For full automation, customers may set up SSO between on-premise AD 204 and IDCS 208 through, e.g., AD federation service ("AD/FS", or some other mechanism that implements SAML federation) so that end users can get access to cloud applications 210, 212, 214, 216,

and on-premise applications 218 with a single corporate password 332.

[0043] Fig. 3 is a block diagram 300 of an example embodiment that includes the same components 202, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 234 as in Fig. 2. However, in the embodiment of Fig. 3, IDCS 202 provides integration with an on-premise IDM 304 such as Oracle IDM. Oracle IDM 304 is a software suite from Oracle Corp. for providing IAM functionality. The embodiment provides seamless user experience across all applications including on-premise and third-party applications. The embodiment provisions user identities from on-premise IDM 304 to IDCS 208 via SCIM identity bus 234 between cloud 202 and enterprise 206. The embodiment further provides SAML bus 228 (or an OpenID Connect bus) for federating authentication from cloud 208 to on-premise 206.

[0044] In the embodiment of Fig. 3, an Oracle Identity Manager ("OIM") Connector 302 from Oracle Corp., and an Oracle Access Manager ("OAM") federation module 306 from Oracle Corp., are implemented as extension modules of Oracle IDM 304. A connector is a module that has physical awareness about how to talk to a system. OIM is an application configured to manage user identities (e.g., manage user accounts in different systems based on what a user should and should not have access to). OAM is a security application that provides access management functionality such as web SSO; identity context, authentication and authorization; policy administration; testing; logging; auditing; etc. OAM has built-in support for SAML. If a user has an account in IDCS 202, OIM connector 302 and OAM federation 306 can be used with Oracle IDM 304 to create/delete that account and manage access from that account.

[0045] Fig. 4 is a block diagram 400 of an example embodiment that includes the same components 202, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 234 as in Figs. 2 and 3. However, in the embodiment of Fig. 3, IDCS 202 provides functionality to extend cloud identities to on-premise applications 218. The embodiment provides seamless view of the identity across all applications including on-premise and third-party applications. In the embodiment of Fig. 4, SCIM identity bus 234 is used to synchronize data in IDCS 202 with on-premise LDAP data called "Cloud Cache" 402. Cloud Cache 402 is disclosed in more detail below.

[0046] Generally, an application that is configured to communicate based on LDAP needs an LDAP connection. An LDAP connection may not be established by such application through a URL (unlike, e.g., "www.google.com" that makes a connection to Google) since the LDAP needs to be on a local network. In the embodiment of Fig. 4, an LDAP-based application 218 makes a connection to Cloud Cache 402, and Cloud Cache 402 establishes a connection to IDCS 202 and then pulls data from IDCS 202 as it is being requested. The communication between IDCS 202 and Cloud Cache 402 may be implemented according to the SCIM protocol. For example, Cloud Cache 402 may use SCIM bus 234 to send a SCIM request to IDCS 202 and receive corresponding data in return.

[0047] Generally, fully implementing an application includes building a consumer portal, running marketing campaigns on the external user population, supporting web and mobile channels, and dealing with user authentication, sessions, user profiles, user groups, application roles, password policies, self-service/registration, social integration,

identity federation, etc. Generally, application developers are not identity/security experts. Therefore, on-demand identity management services are desired.

[0048] Fig. 5 is a block diagram 500 of an example embodiment that includes the same components 202, 220, 222, 224, 226, 234, 402, as in Figs. 2-4. However, in the embodiment of Fig. 5, IDCS 202 provides secure identity management on demand. The embodiment provides on demand integration with identity services of IDCS 202 (e.g., based on standards such as OpenID Connect, OAuth2, SAML2, or SCIM). Applications 505 (which may be on-premise, in a public cloud, or in a private cloud) may call identity service APIs 504 in IDCS 202. The services provided by IDCS 202 may include, for example, self-service registration 506, password management 508, user profile management 510, user authentication 512, token management 514, social integration 516, etc.

[0049] In this embodiment, SCIM identity bus 234 is used to synchronize data in IDCS 202 with data in on-premise LDAP Cloud Cache 402. Further, a "Cloud Gate" 502 running on a web server/proxy (e.g., NGINX, Apache, etc.) may be used by applications 505 to obtain user web SSO and REST API security from IDCS 202. Cloud Gate 502 is a component that secures access to multi-tenant IDCS microservices by ensuring that client applications provide valid access tokens, and/or users successfully authenticate in order to establish SSO sessions. Cloud Gate 502 is further disclosed below. Cloud Gate 502 (enforcement point similar to webgate/webagent) enables applications running behind supported web servers to participate in SSO.

[0050] One embodiment provides SSO and cloud SSO functionality. A general

point of entry for both on-premise IAM and IDCS in many organizations is SSO. Cloud SSO enables users to access multiple cloud resources with a single user sign-in. Often, organizations will want to federate their on-premise identities. Accordingly, embodiments utilize open standards to allow for integration with existing SSO to preserve and extend investment (e.g., until a complete, eventual transition to an identity cloud service approach is made).

[0051] One embodiment may provide the following functionalities:

- maintain an identity store to track user accounts, ownership, access, and permissions that have been authorized,
- integrate with workflow to facilitate various approvals (e.g., management, IT, human resources, legal, and compliance) needed for applications access,
- provision SaaS user accounts for selective devices (e.g., mobile and personal computer ("PC")) with access to user portal containing many private and public cloud resources, and
- facilitate periodic management attestation review for compliance with regulations and current job responsibilities.

[0052] In addition to these functions, embodiments may further provide:

- cloud account provisioning to manage account life cycle in cloud applications,
- more robust multifactor authentication ("MFA") integration,
- extensive mobile security capabilities, and
- dynamic authentication options.

[0053] One embodiment provides adaptive authentication and MFA. Generally,

passwords and challenge questions have been seen as inadequate and susceptible to common attacks such as phishing. Most business entities today are looking at some form of MFA to reduce risk. To be successfully deployed, however, solutions need to be easily provisioned, maintained, and understood by the end user, as end users usually resist anything that interferes with their digital experience. Companies are looking for ways to securely incorporate bring your own device ("BYOD"), social identities, remote users, customers, and contractors, while making MFA an almost transparent component of a seamless user access experience. Within an MFA deployment, industry standards such as OAuth and OpenID Connect are essential to ensure integration of existing multifactor solutions and the incorporation of newer, adaptive authentication technology. Accordingly, embodiments define dynamic (or adaptive) authentication as the evaluation of available information (i.e., IP address, location, time of day, and biometrics) to prove an identity after a user session has been initiated. With the appropriate standards (e.g., open authentication ("OATH") and fast identity online ("FIDO")) integration and extensible identity management framework, embodiments provide MFA solutions that can be adopted, upgraded, and integrated easily within an IT organization as part of an end-to-end secure IAM deployment. When considering MFA and adaptive policies, organizations must implement consistent policies across on-premise and cloud resources, which in a hybrid IDCS and on-premise IAM environment requires integration between systems.

[0054] One embodiment provides user provisioning and certification. Generally, the fundamental function of an IAM solution is to enable and support the entire user

provisioning life cycle. This includes providing users with the application access appropriate for their identity and role within the organization, certifying that they have the correct ongoing access permissions (e.g., as their role or the tasks or applications used within their role change over time), and promptly de-provisioning them as their departure from the organization may require. This is important not only for meeting various compliance requirements but also because inappropriate insider access is a major source of security breaches and attacks. An automated user provisioning capability within an identity cloud solution can be important not only in its own right but also as part of a hybrid IAM solution whereby IDCS provisioning may provide greater flexibility than an on-premise solution for transitions as a company downsizes, upsizes, merges, or looks to integrate existing systems with IaaS/PaaS/SaaS environments. An IDCS approach can save time and effort in one-off upgrades and ensure appropriate integration among necessary departments, divisions, and systems. The need to scale this technology often sneaks up on corporations, and the ability to deliver a scalable IDCS capability immediately across the enterprise can provide benefits in flexibility, cost, and control.

[0055] Generally, an employee is granted additional privileges (i.e., "privilege creep") over the years as her/his job changes. Companies that are lightly regulated generally lack an "attestation" process that requires managers to regularly audit their employees' privileges (e.g., access to networks, servers, applications, and data) to halt or slow the privilege creep that results in over-privileged accounts. Accordingly, one embodiment may provide a regularly conducted (at least once a year) attestation

process. Further, with mergers and acquisitions, the need for these tools and services increases exponentially as users are on SaaS systems, on-premise, span different departments, and/or are being de-provisioned or re-allocated. The move to cloud can further complicate this situation, and the process can quickly escalate beyond existing, often manually managed, certification methods. Accordingly, one embodiment automates these functions and applies sophisticated analytics to user profiles, access history, provisioning/de-provisioning, and fine-grained entitlements.

[0056] One embodiment provides identity analytics. Generally, the ability to integrate identity analytics with the IAM engine for comprehensive certification and attestation can be critical to securing an organization's risk profile. Properly deployed identity analytics can demand total internal policy enforcement. Identity analytics that provide a unified single management view across cloud and on-premise are much needed in a proactive governance, risk, and compliance ("GRC") enterprise environment, and can aid in providing a closed-loop process for reducing risk and meeting compliance regulations. Accordingly, one embodiment provides identity analytics that are easily customizable by the client to accommodate specific industry demands and government regulations for reports and analysis required by managers, executives, and auditors.

[0057] One embodiment provides self-service and access request functionality to improve the experience and efficiency of the end user and to reduce costs from help desk calls. Generally, while a number of companies deploy on-premise self-service access request for their employees, many have not extended these systems adequately

outside the formal corporate walls. Beyond employee use, a positive digital customer experience increases business credibility and ultimately contributes to revenue increase, and companies not only save on customer help desk calls and costs but also improve customer satisfaction. Accordingly, one embodiment provides an identity cloud service environment that is based on open standards and seamlessly integrates with existing access control software and MFA mechanisms when necessary. The SaaS delivery model saves time and effort formerly devoted to systems upgrades and maintenance, freeing professional IT staff to focus on more core business applications.

[0058] One embodiment provides privileged account management ("PAM"). Generally, every organization, whether using SaaS, PaaS, IaaS, or on-premise applications, is vulnerable to unauthorized privileged account abuse by insiders with super-user access credentials such as system administrators, executives, HR officers, contractors, systems integrators, etc. Moreover, outside threats typically first breach a low-level user account to eventually reach and exploit privileged user access controls within the enterprise system. Accordingly, one embodiment provides PAM to prevent such unauthorized insider account use. The main component of a PAM solution is a password vault which may be delivered in various ways, e.g., as software to be installed on an enterprise server, as a virtual appliance also on an enterprise server, as a packaged hardware/software appliance, or as part of a cloud service. PAM functionality is similar to a physical safe used to store passwords kept in an envelope and changed periodically, with a manifest for signing them in and out. One embodiment allows for a password checkout as well as setting time limits, forcing periodic changes, automatically

tracking checkout, and reporting on all activities. One embodiment provides a way to connect directly through to a requested resource without the user ever knowing the password. This capability also paves the way for session management and additional functionality.

[0059] Generally, most cloud services utilize APIs and administrative interfaces, which provide opportunities for infiltrators to circumvent security. Accordingly, one embodiment accounts for these holes in PAM practices as the move to the cloud presents new challenges for PAM. Many small to medium sized businesses now administer their own SaaS systems (e.g., Office 365), while larger companies increasingly have individual business units spinning up their own SaaS and IaaS services. These customers find themselves with PAM capabilities within the identity cloud service solutions or from their IaaS/PaaS provider but with little experience in handling this responsibility. Moreover, in some cases, many different geographically dispersed business units are trying to segregate administrative responsibilities for the same SaaS applications. Accordingly, one embodiment allows customers in these situations to link existing PAM into the overall identity framework of the identity cloud service and move toward greater security and compliance with the assurance of scaling to cloud load requirements as business needs dictate.

API Platform

[0060] Embodiments provide an API platform that exposes a collection of capabilities as services. The APIs are aggregated into microservices and each microservice exposes one or more of the APIs. That is, each microservice may expose

different types of APIs. In one embodiment, each microservice communicates only through its APIs. In one embodiment, each API may be a microservice. In one embodiment, multiple APIs are aggregated into a service based on a target capability to be provided by that service (e.g., OAuth, SAML, Admin, etc.). As a result, similar APIs are not exposed as separate runtime processes. The APIs are what is made available to a service consumer to use the services provided by IDCS.

[0061] Generally, in the web environment of IDCS, a URL includes three parts: a host, a microservice, and a resource (e.g., host/microservice/resource). In one embodiment, the microservice is characterized by having a specific URL prefix, e.g., "host/oauth/v1" where the actual microservice is "oauth/v1", and under "oauth/v1" there are multiple APIs, e.g., an API to request tokens: "host/oauth/v1/token", an API to authenticate a user: "host/oauth/v1/authorize", etc. That is, the URL implements a microservice, and the resource portion of the URL implements an API. Accordingly, multiple APIs are aggregated under the same microservice. In one embodiment, the host portion of the URL identifies a tenant (e.g., https://tenant3.identity.oraclecloud.com:/oauth/v1/token").

[0062] Configuring applications that integrate with external services with the necessary endpoints and keeping that configuration up to date is typically a challenge. To meet this challenge, embodiments expose a public discovery API at a well-known location from where applications can discover the information about IDCS they need in order to consume IDCS APIs. In one embodiment, two discovery documents are supported: IDCS Configuration (which includes IDCS, SAML, SCIM, OAuth, and

OpenID Connect configuration, at e.g., <IDCS-URL>/.well-known/idcs-configuration), and Industry-standard OpenID Connect Configuration (at, e.g., <IDCS-URL>/.well-known/openid-configuration). Applications can retrieve discovery documents by being configured with a single IDCS URL.

[0063] Fig. 6 is a block diagram providing a system view 600 of IDCS in one embodiment. In Fig. 6, any one of a variety of applications/services 602 may make HTTP calls to IDCS APIs to use IDCS services. Examples of such applications/services 602 are web applications, native applications (e.g., applications that are built to run on a specific operating system, such as Windows applications, iOS applications, Android applications, etc.), web services, customer applications, partner applications, or any services provided by a public cloud, such as Software as a Service ("SaaS"), PaaS, and Infrastructure as a Service ("IaaS").

[0064] In one embodiment, the HTTP requests of applications/services 602 that require IDCS services go through an Oracle Public Cloud BIG-IP appliance 604 and an IDCS BIG-IP appliance 606 (or similar technologies such as a Load Balancer, or a component called a Cloud Load Balancer as a Service ("LBaaS") that implements appropriate security rules to protect the traffic). However, the requests can be received in any manner. At IDCS BIG-IP appliance 606 (or, as applicable, a similar technology such as a Load Balancer or a Cloud LBaaS), a cloud provisioning engine 608 performs tenant and service orchestration. In one embodiment, cloud provisioning engine 608 manages internal security artifacts associated with a new tenant being on-boarded into the cloud or a new service instance purchased by a customer.

[0065] The HTTP requests are then received by an IDCS web routing tier 610 that implements a security gate (i.e., Cloud Gate) and provides service routing and microservices registration and discovery 612. Depending on the service requested, the HTTP request is forwarded to an IDCS microservice in the IDCS middle tier 614. IDCS microservices process external and internal HTTP requests. IDCS microservices implement platform services and infrastructure services. IDCS platform services are separately deployed Java-based runtime services implementing the business of IDCS. IDCS infrastructure services are separately deployed runtime services providing infrastructure support for IDCS. IDCS further includes infrastructure libraries that are common code packaged as shared libraries used by IDCS services and shared libraries. Infrastructure services and libraries provide supporting capabilities as required by platform services for implementing their functionality.

Platform Services

[0066] In one embodiment, IDCS supports standard authentication protocols, hence IDCS microservices include platform services such as OpenID Connect, OAuth, SAML2, System for Cross-domain Identity Management++ ("SCIM++"), etc.

[0067] The OpenID Connect platform service implements standard OpenID Connect Login/Logout flows. Interactive web-based and native applications leverage standard browser-based OpenID Connect flow to request user authentication, receiving standard identity tokens that are JavaScript Object Notation ("JSON") Web Tokens ("JWTs") conveying the user's authenticated identity. Internally, the runtime authentication model is stateless, maintaining the user's authentication/session state in

the form of a host HTTP cookie (including the JWT identity token). The authentication interaction initiated via the OpenID Connect protocol is delegated to a trusted SSO service that implements the user login/logout ceremonies for local and federated logins. Further details of this functionality are disclosed below with reference to Figs. 10 and 11. In one embodiment, OpenID Connect functionality is implemented according to, for example, OpenID Foundation standards.

[0068] The OAuth2 platform service provides token authorization services. It provides a rich API infrastructure for creating and validating access tokens conveying user rights to make API calls. It supports a range of useful token grant types, enabling customers to securely connect clients to their services. It implements standard 2-legged and 3-legged OAuth2 token grant types. Support for OpenID Connect ("OIDC") enables compliant applications (OIDC relaying parties ("RP"s)) to integrate with IDCS as the identity provider (OIDC OpenID provider ("OP")). Similarly, the integration of IDCS as OIDC RP with social OIDC OP (e.g., Facebook, Google, etc.) enables customers to allow social identities policy-based access to applications. In one embodiment, OAuth functionality is implemented according to, for example, Internet Engineering Task Force ("IETF"), Request for Comments ("RFC") 6749.

[0069] The SAML2 platform service provides identity federation services. It enables customers to set up federation agreements with their partners based on SAML identity provider ("IDP") and SAML service provider ("SP") relationship models. In one embodiment, the SAML2 platform service implements standard SAML2 Browser POST Login and Logout Profiles. In one embodiment, SAML functionality is implemented

according to, for example, IETF, RFC 7522.

[0070] SCIM is an open standard for automating the exchange of user identity information between identity domains or information technology ("IT") systems, as provided by, e.g., IETF, RFCs 7642, 7643, 7644. The SCIM++ platform service provides identity administration services and enables customers to access IDP features of IDCS. The administration services expose a set of stateless REST interfaces (i.e., APIs) that cover identity lifecycle, password management, group management, etc., exposing such artifacts as web-accessible resources.

[0071] All IDCS configuration artifacts are resources, and the APIs of the administration services allow for managing IDCS resources (e.g., users, roles, password policies, applications, SAML/OIDC identity providers, SAML service providers, keys, certifications, notification templates, etc.). Administration services leverage and extend the SCIM standard to implement schema-based REST APIs for Create, Read, Update, Delete, and Query ("CRUDQ") operations on all IDCS resources. Additionally, all internal resources of IDCS used for administration and configuration of IDCS itself are exposed as SCIM-based REST APIs. Access to the identity store 618 is isolated to the SCIM++ API.

[0072] In one embodiment, for example, the SCIM standard is implemented to manage the users and groups resources as defined by the SCIM specifications, while SCIM++ is configured to support additional IDCS internal resources (e.g., password policies, roles, settings, etc.) using the language defined by the SCIM standard.

[0073] The Administration service supports the SCIM 2.0 standard endpoints

with the standard SCIM 2.0 core schemas and schema extensions where needed. In addition, the Administration service supports several SCIM 2.0 compliant endpoint extensions to manage other IDCS resources, for example, Users, Groups, Applications, Settings, etc. The Administration service also supports a set of remote procedure call-style ("RPC-style") REST interfaces that do not perform CRUDQ operations but instead provide a functional service, for example, "UserPasswordGenerator," "UserPasswordValidator," etc.

[0074] IDCS Administration APIs use the OAuth2 protocol for authentication and authorization. IDCS supports common OAuth2 scenarios such as scenarios for web server, mobile, and JavaScript applications. Access to IDCS APIs is protected by access tokens. To access IDCS Administration APIs, an application needs to be registered as an OAuth2 client or an IDCS Application (in which case the OAuth2 client is created automatically) through the IDCS Administration console and be granted desired IDCS Administration Roles. When making IDCS Administration API calls, the application first requests an access token from the IDCS OAuth2 Service. After acquiring the token, the application sends the access token to the IDCS API by including it in the HTTP authorization header. Applications can use IDCS Administration REST APIs directly, or use an IDCS Java Client API Library.

Infrastructure Services

[0075] The IDCS infrastructure services support the functionality of IDCS platform services. These runtime services include an event processing service (for asynchronously processing user notifications, application subscriptions, and auditing to

database); a job scheduler service (for scheduling and executing jobs, e.g., executing immediately or at a configured time long-running tasks that do not require user intervention); a cache management service; a storage management service (for integrating with a public cloud storage service); a reports service (for generating reports and dashboards); an SSO service (for managing internal user authentication and SSO); a user interface ("UI") service (for hosting different types of UI clients); and a service manager service. Service manager is an internal interface between the Oracle Public Cloud and IDCS. Service manager manages commands issued by the Oracle Public Cloud, where the commands need to be implemented by IDCS. For example, when a customer signs up for an account in a cloud store before they can buy something, the cloud sends a request to IDCS asking to create a tenant. In this case, service manager implements the cloud specific operations that the cloud expects IDCS to support.

[0076] An IDCS microservice may call another IDCS microservice through a network interface (i.e., an HTTP request).

[0077] In one embodiment, IDCS may also provide a schema service (or a persistence service) that allows for using a database schema. A schema service allows for delegating the responsibility of managing database schemas to IDCS. Accordingly, a user of IDCS does not need to manage a database since there is an IDCS service that provides that functionality. For example, the user may use the database to persist schemas on a per tenant basis, and when there is no more space in the database, the schema service will manage the functionality of obtaining another database and growing the space so that the users do not have to manage the database themselves.

[0078] IDCS further includes data stores which are data repositories required/generated by IDCS, including an identity store 618 (storing users, groups, etc.), a global database 620 (storing configuration data used by IDCS to configure itself), an operational schema 622 (providing per tenant schema separation and storing customer data on a per customer basis), an audit schema 624 (storing audit data), a caching cluster 626 (storing cached objects to speed up performance), etc. All internal and external IDCS consumers integrate with the identity services over standards-based protocols. This enables use of a domain name system ("DNS") to resolve where to route requests, and decouples consuming applications from understanding the internal implementation of identity services.

Real-Time and Near-Real-Time Tasks

[0079] IDCS separates the tasks of a requested service into synchronous real-time and asynchronous near-real-time tasks, where real-time tasks include only the operations that are needed for the user to proceed. In one embodiment, a real-time task is a task that is performed with minimal delay, and a near-real-time task is a task that is performed in the background without the user having to wait for it. In one embodiment, a real-time task is a task that is performed with substantially no delay or with negligible delay, and appears to a user as being performed almost instantaneously.

[0080] The real-time tasks perform the main business functionality of a specific identity service. For example, when requesting a login service, an application sends a message to authenticate a user's credentials and get a session cookie in return. What the user experiences is logging into the system. However, several other tasks may be

performed in connection with the user's logging in, such as validating who the user is, auditing, sending notifications, etc. Accordingly, validating the credentials is a task that is performed in real-time so that the user is given an HTTP cookie to start a session, but the tasks related to notifications (e.g., sending an email to notify the creation of an account), audits (e.g., tracking/recording), etc., are near-real-time tasks that can be performed asynchronously so that the user can proceed with least delay.

[0081] When an HTTP request for a microservice is received, the corresponding real-time tasks are performed by the microservice in the middle tier, and the remaining near-real-time tasks such as operational logic/events that are not necessarily subject to real-time processing are offloaded to message queues 628 that support a highly scalable asynchronous event management system 630 with guaranteed delivery and processing. Accordingly, certain behaviors are pushed from the front end to the backend to enable IDCS to provide high level service to the customers by reducing latencies in response times. For example, a login process may include validation of credentials, submission of a log report, updating of the last login time, etc., but these tasks can be offloaded to a message queue and performed in near-real-time as opposed to real-time.

[0082] In one example, a system may need to register or create a new user. The system calls an IDCS SCIM API to create a user. The end result is that when the user is created in identity store 618, the user gets a notification email including a link to reset their password. When IDCS receives a request to register or create a new user, the corresponding microservice looks at configuration data in the operational database

(located in global database 620 in Fig. 6) and determines that the "create user" operation is marked with a "create user" event which is identified in the configuration data as an asynchronous operation. The microservice returns to the client and indicates that the creation of the user is done successfully, but the actual sending of the notification email is postponed and pushed to the backend. In order to do so, the microservice uses a messaging API 616 to queue the message in queue 628 which is a store.

[0083] In order to dequeue queue 628, a messaging microservice, which is an infrastructure microservice, continually runs in the background and scans queue 628 looking for events in queue 628. The events in queue 628 are processed by event subscribers 630 such as audit, user notification, application subscriptions, data analytics, etc. Depending on the task indicated by an event, event subscribers 630 may communicate with, for example, audit schema 624, a user notification service 634, an identity event subscriber 632, etc. For example, when the messaging microservice finds the "create user" event in queue 628, it executes the corresponding notification logic and sends the corresponding email to the user.

[0084] In one embodiment, queue 628 queues operational events published by microservices 614 as well as resource events published by APIs 616 that manage IDCS resources.

[0085] IDCS uses a real-time caching structure to enhance system performance and user experience. The cache itself may also be provided as a microservice. IDCS implements an elastic cache cluster 626 that grows as the number of customers

supported by IDCS scales. Cache cluster 626 may be implemented with a distributed data grid which is disclosed in more detail below. In one embodiment, write-only resources bypass cache.

[0086] In one embodiment, IDCS runtime components publish health and operational metrics to a public cloud monitoring module 636 that collects such metrics of a public cloud such as Oracle Public Cloud from Oracle Corp.

[0087] In one embodiment, IDCS may be used to create a user. For example, a client application 602 may issue a REST API call to create a user. Admin service (a platform service in 614) delegates the call to a user manager (an infrastructure library/service in 614), which in turn creates the user in the tenant-specific ID store stripe in ID store 618. On "User Create Success", the user manager audits the operation to the audit table in audit schema 624, and publishes an "identity.user.create.success" event to message queue 628. Identity subscriber 632 picks up the event and sends a "Welcome" email to the newly created user, including newly created login details.

[0088] In one embodiment, IDCS may be used to grant a role to a user, resulting in a user provisioning action. For example, a client application 602 may issue a REST API call to grant a user a role. Admin service (a platform service in 614) delegates the call to a role manager (an infrastructure library/service in 614), who grants the user a role in the tenant-specific ID store stripe in ID store 618. On "Role Grant Success", the role manager audits the operations to the audit table in audit schema 624, and publishes an "identity.user.role.grant.success" event to message queue 628. Identity

subscriber 632 picks up the event and evaluates the provisioning grant policy. If there is an active application grant on the role being granted, a provisioning subscriber performs some validation, initiates account creation, calls out the target system, creates an account on the target system, and marks the account creation as successful. Each of these functionalities may result in publishing of corresponding events, such as "prov.account.create.initiate", "prov.target.create.initiate", "prov.target.create.success", or "prov.account.create.success". These events may have their own business metrics aggregating number of accounts created in the target system over the last N days.

[0089] In one embodiment, IDCS may be used for a user to log in. For example, a client application 602 may use one of the supported authentication flows to request a login for a user. IDCS authenticates the user, and upon success, audits the operation to the audit table in audit schema 624. Upon failure, IDCS audits the failure in audit schema 624, and publishes "login.user.login.failure" event in message queue 628. A login subscriber picks up the event, updates its metrics for the user, and determines if additional analytics on the user's access history need to be performed.

[0090] Accordingly, by implementing "inversion of control" functionality (e.g., changing the flow of execution to schedule the execution of an operation at a later time so that the operation is under the control of another system), embodiments enable additional event queues and subscribers to be added dynamically to test new features on a small user sample before deploying to broader user base, or to process specific events for specific internal or external customers.

Stateless Functionality

[0091] IDCS microservices are stateless, meaning the microservices themselves do not maintain state. "State" refers to the data that an application uses in order to perform its capabilities. IDCS provides multi-tenant functionality by persisting all state into tenant specific repositories in the IDCS data tier. The middle tier (i.e., the code that processes the requests) does not have data stored in the same location as the application code. Accordingly, IDCS is highly scalable, both horizontally and vertically.

[0092] To scale vertically (or scale up/down) means to add resources to (or remove resources from) a single node in a system, typically involving the addition of CPUs or memory to a single computer. Vertical scalability allows an application to scale up to the limits of its hardware. To scale horizontally (or scale out/in) means to add more nodes to (or remove nodes from) a system, such as adding a new computer to a distributed software application. Horizontal scalability allows an application to scale almost infinitely, bound only by the amount of bandwidth provided by the network.

[0093] Stateless-ness of the middle tier of IDCS makes it horizontally scalable just by adding more CPUs, and the IDCS components that perform the work of the application do not need to have a designated physical infrastructure where a particular application is running. Stateless-ness of the IDCS middle tier makes IDCS highly available, even when providing identity services to a very large number of customers/tenants. Each pass through an IDCS application/service is focused on CPU usage only to perform the application transaction itself but not use hardware to store data. Scaling is accomplished by adding more slices when the application is running, while data for the transaction is stored at a persistence layer where more copies can be

added when needed.

[0094] The IDCS web tier, middle tier, and data tier can each scale independently and separately. The web tier can be scaled to handle more HTTP requests. The middle tier can be scaled to support more service functionality. The data tier can be scaled to support more tenants.

IDCS Functional View

[0095] Fig. 6A is an example block diagram 600b of a functional view of IDCS in one embodiment. In block diagram 600b, the IDCS functional stack includes services, shared libraries, and data stores. The services include IDCS platform services 640b, IDCS premium services 650b, and IDCS infrastructure services 662b. In one embodiment, IDCS platform services 640b and IDCS premium services 650b are separately deployed Java-based runtime services implementing the business of IDCS, and IDCS infrastructure services 662b are separately deployed runtime services providing infrastructure support for IDCS. The shared libraries include IDCS infrastructure libraries 680b which are common code packaged as shared libraries used by IDCS services and shared libraries. The data stores are data repositories required/generated by IDCS, including identity store 698b, global configuration 700b, message store 702b, global tenant 704b, personalization settings 706b, resources 708b, user transient data 710b, system transient data 712b, per-tenant schemas (managed ExaData) 714b, operational store (not shown), caching store (not shown), etc.

[0096] In one embodiment, IDCS platform services 640b include, for example,

OpenID Connect service 642b, OAuth2 service 644b, SAML2 service 646b, and SCIM++ service 648b. In one embodiment, IDCS premium services include, for example, cloud SSO and governance 652b, enterprise governance 654b, AuthN broker 656b, federation broker 658b, and private account management 660b.

[0097] IDCS infrastructure services 662b and IDCS infrastructure libraries 680b provide supporting capabilities as required by IDCS platform services 640b to do their work. In one embodiment, IDCS infrastructure services 662b include job scheduler 664b, UI 666b, SSO 668b, reports 670b, cache 672b, storage 674b, service manager 676b (public cloud control), and event processor 678b (user notifications, app subscriptions, auditing, data analytics). In one embodiment, IDCS infrastructure libraries 680b include data manager APIs 682b, event APIs 684b, storage APIs 686b, authentication APIs 688b, authorization APIs 690b, cookie APIs 692b, keys APIs 694b, and credentials APIs 696b. In one embodiment, cloud compute service 602b (internal Nimbula) supports the function of IDCS infrastructure services 662b and IDCS infrastructure libraries 680b.

[0098] In one embodiment, IDCS provides various UIs 602b for a consumer of IDCS services, such as customer end user UI 604b, customer admin UI 606b, DevOps admin UI 608b, and login UI 610b. In one embodiment, IDCS allows for integration 612b of applications (e.g., customer apps 614b, partner apps 616b, and cloud apps 618b) and firmware integration 620b. In one embodiment, various environments may integrate with IDCS to support their access control needs. Such integration may be provided by, for example, identity bridge 622b (providing AD integration, WNA, and

SCIM connector), Apache agent 624b, or MSFT agent 626b.

[0099] In one embodiment, internal and external IDCS consumers integrate with the identity services of IDCS over standards-based protocols 628b, such as OpenID Connect 630b, OAuth2 632b, SAML2 634b, SCIM 636b, and REST/HTTP 638b. This enables use of a domain name system ("DNS") to resolve where to route requests, and decouples the consuming applications from understanding internal implementation of the identity services.

[00100] The IDCS functional view in Fig. 6A further includes public cloud infrastructure services that provide common functionality that IDCS depends on for user notifications (cloud notification service 718b), file storage (cloud storage service 716b), and metrics/alerting for DevOps (cloud monitoring service (EM) 722b and cloud metrics service (Graphite) 720b).

Cloud Gate

[00101] In one embodiment, IDCS implements a "Cloud Gate" in the web tier. Cloud Gate is a web server plugin that enables web applications to externalize user SSO to an identity management system (e.g., IDCS), similar to WebGate or WebAgent technologies that work with enterprise IDM stacks. Cloud Gate acts as a security gatekeeper that secures access to IDCS APIs. In one embodiment, Cloud Gate is implemented by a web/proxy server plugin that provides a web Policy Enforcement Point ("PEP") for protecting HTTP resources based on OAuth.

[00102] Fig. 7 is a block diagram 700 of an embodiment that implements a Cloud Gate 702 running in a web server 712 and acting as a Policy Enforcement Point ("PEP")

configured to integrate with IDCS Policy Decision Point ("PDP") using open standards (e.g., OAuth2, OpenID Connect, etc.) while securing access to web browser and REST API resources 714 of an application. In some embodiments, the PDP is implemented at OAuth and/or OpenID Connect microservices 704. For example, when a user browser 706 sends a request to IDCS for a login of a user 710, a corresponding IDCS PDP validates the credentials and then decides whether the credentials are sufficient (e.g., whether to request for further credentials such as a second password). In the embodiment of Fig. 7, Cloud Gate 702 may act both as the PEP and as the PDP since it has a local policy.

[00103] As part of one-time deployment, Cloud Gate 702 is registered with IDCS as an OAuth2 client, enabling it to request OIDC and OAuth2 operations against IDCS. Thereafter, it maintains configuration information about an application's protected and unprotected resources, subject to request matching rules (how to match URLs, e.g., with wild cards, regular expressions, etc.). Cloud Gate 702 can be deployed to protect different applications having different security policies, and the protected applications can be multi-tenant.

[00104] During web browser-based user access, Cloud Gate 702 acts as an OIDC RP 718 initiating a user authentication flow. If user 710 has no valid local user session, Cloud Gate 702 re-directs the user to the SSO microservice and participates in the OIDC "Authorization Code" flow with the SSO microservice. The flow concludes with the delivery of a JWT as an identity token. Cloud Gate 708 validates the JWT (e.g., looks at signature, expiration, destination/audience, etc.) and issues a local session

cookie for user 710. It acts as a session manager 716 securing web browser access to protected resources and issuing, updating, and validating the local session cookie. It also provides a logout URL for removal of its local session cookie.

[00105] Cloud Gate 702 also acts as an HTTP Basic Auth authenticator, validating HTTP Basic Auth credentials against IDCS. This behavior is supported in both session-less and session-based (local session cookie) modes. No server-side IDCS session is created in this case.

[00106] During programmatic access by REST API clients 708, Cloud Gate 702 may act as an OAuth2 resource server/filter 720 for an application's protected REST APIs 714. It checks for the presence of a request with an authorization header and an access token. When client 708 (e.g., mobile, web apps, JavaScript, etc.) presents an access token (issued by IDCS) to use with a protected REST API 714, Cloud Gate 702 validates the access token before allowing access to the API (e.g., signature, expiration, audience, etc.). The original access token is passed along unmodified.

[00107] Generally, OAuth is used to generate either a client identity propagation token (e.g., indicating who the client is) or a user identity propagation token (e.g., indicating who the user is). In the embodiments, the implementation of OAuth in Cloud Gate is based on a JWT which defines a format for web tokens, as provided by, e.g., IETF, RFC 7519.

[00108] When a user logs in, a JWT is issued. The JWT is signed by IDCS and supports multi-tenant functionality in IDCS. Cloud Gate validates the JWT issued by IDCS to allow for multi-tenant functionality in IDCS. Accordingly, IDCS provides multi-

tenancy in the physical structure as well as in the logical business process that underpins the security model.

Tenancy Types

[00109] IDCS specifies three types of tenancies: customer tenancy, client tenancy, and user tenancy. Customer or resource tenancy specifies who the customer of IDCS is (i.e., for whom is the work being performed). Client tenancy specifies which client application is trying to access data (i.e., what application is doing the work). User tenancy specifies which user is using the application to access data (i.e., by whom is the work being performed). For example, when a professional services company provides system integration functionality for a warehouse club and uses IDCS for providing identity management for the warehouse club systems, user tenancy corresponds to the professional services company, client tenancy is the application that is used to provide system integration functionality, and customer tenancy is the warehouse club.

[00110] Separation and identification of these three tenancies enables multi-tenant functionality in a cloud-based service. Generally, for on-premise software that is installed on a physical machine on-premise, there is no need to specify three different tenancies since a user needs to be physically on the machine to log in. However, in a cloud-based service structure, embodiments use tokens to determine who is using what application to access which resources. The three tenancies are codified by tokens, enforced by Cloud Gate, and used by the business services in the middle tier. In one embodiment, an OAuth server generates the tokens. In various embodiments, the

tokens may be used in conjunction with any security protocol other than OAuth.

[00111] Decoupling user, client, and resource tenancies provides substantial business advantages for the users of the services provided by IDCS. For example, it allows a service provider that understands the needs of a business (e.g., a healthcare business) and their identity management problems to buy services provided by IDCS, develop their own backend application that consumes the services of IDCS, and provide the backend applications to the target businesses. Accordingly, the service provider may extend the services of IDCS to provide their desired capabilities and offer those to certain target businesses. The service provider does not have to build and run software to provide identity services but can instead extend and customize the services of IDCS to suit the needs of the target businesses.

[00112] Some known systems only account for a single tenancy which is customer tenancy. However, such systems are inadequate when dealing with access by a combination of users such as customer users, customer's partners, customer's clients, clients themselves, or clients that customer has delegated access to. Defining and enforcing multiple tenancies in the embodiments facilitates the identity management functionality over such variety of users.

[00113] In one embodiment, one entity of IDCS does not belong to multiple tenants at the same time; it belongs to only one tenant, and a "tenancy" is where artifacts live. Generally, there are multiple components that implement certain functions, and these components can belong to tenants or they can belong to infrastructure. When infrastructure needs to act on behalf of tenants, it interacts with an

entity service on behalf of the tenant. In that case, infrastructure itself has its own tenancy and customer has its own tenancy. When a request is submitted, there can be multiple tenancies involved in the request.

[00114] For example, a client that belongs to "tenant 1" may execute a request to get a token for "tenant 2" specifying a user in "tenant 3." As another example, a user living in "tenant 1" may need to perform an action in an application owned by "tenant 2". Thus, the user needs to go to the resource namespace of "tenant 2" and request a token for themselves. Accordingly, delegation of authority is accomplished by identifying "who" can do "what" to "whom." As yet another example, a first user working for a first organization ("tenant 1") may allow a second user working for a second organization ("tenant 2") to have access to a document hosted by a third organization ("tenant 3").

[00115] In one example, a client in "tenant 1" may request an access token for a user in "tenant 2" to access an application in "tenant 3". The client may do so by invoking an OAuth request for the token by going to "http://tenant3/oauth/token". The client identifies itself as a client that lives in "tenant 1" by including a "client assertion" in the request. The client assertion includes a client ID (e.g., "client 1") and the client tenancy "tenant 1". As "client 1" in "tenant 1", the client has the right to invoke a request for a token on "tenant 3", and the client wants the token for a user in "tenant 2". Accordingly, a "user assertion" is also passed as part of the same HTTP request. The access token that is generated will be issued in the context of the target tenancy which is the application tenancy ("tenant 3") and will include the user tenancy ("tenant 2").

[00116] In one embodiment, in the data tier, each tenant is implemented as a separate stripe. From a data management perspective, artifacts live in a tenant. From a service perspective, a service knows how to work with different tenants, and the multiple tenancies are different dimensions in the business function of a service. Fig. 8 illustrates an example system 800 implementing multiple tenancies in an embodiment. System 800 includes a client 802 that requests a service provided by a microservice 804 that understands how to work with data in a database 806. The database includes multiple tenants 808 and each tenant includes the artifacts of the corresponding tenancy. In one embodiment, microservice 804 is an OAuth microservice requested through https://tenant3/oauth/token for getting a token. The function of the OAuth microservice is performed in microservice 804 using data from database 806 to verify that the request of client 802 is legitimate, and if it is legitimate, use the data from different tenancies 808 to construct the token. Accordingly, system 800 is multi-tenant in that it can work in a cross-tenant environment by not only supporting services coming into each tenancy, but also supporting services that can act on behalf of different tenants.

[00117] System 800 is advantageous since microservice 804 is physically decoupled from the data in database 806, and by replicating the data across locations that are closer to the client, microservice 804 can be provided as a local service to the clients and system 800 can manage the availability of the service and provide it globally.

[00118] In one embodiment, microservice 804 is stateless, meaning that the machine that runs microservice 804 does not maintain any markers pointing the service

to any specific tenants. Instead, a tenancy may be marked, for example, on the host portion of a URL of a request that comes in. That tenancy points to one of tenants 808 in database 806. When supporting a large number of tenants (e.g., millions of tenants), microservice 804 cannot have the same number of connections to database 806, but instead uses a connection pool 810 which provides the actual physical connections to database 806 in the context of a database user.

[00119] Generally, connections are built by supplying an underlying driver or provider with a connection string, which is used to address a specific database or server and to provide instance and user authentication credentials (e.g., "Server=sql_box;Database=Common;User ID=uid;Pwd=password;"). Once a connection has been built, it can be opened and closed, and properties (e.g., the command time-out length, or transaction, if one exists) can be set. The connection string includes a set of key-value pairs, dictated by the data access interface of the data provider. A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to a database are required. In connection pooling, after a connection is created, it is placed in the pool and it is used again so that a new connection does not have to be established. For example, when there needs to be ten connections between microservice 804 and database 808, there will be ten open connections in connection pool 810, all in the context of a database user (e.g., in association with a specific database user, e.g., who is the owner of that connection, whose credentials are being validated, is it a database user, is it a system credential, etc.).

[00120] The connections in connection pool 810 are created for a system user that can access anything. Therefore, in order to correctly handle auditing and privileges by microservice 804 processing requests on behalf of a tenant, the database operation is performed in the context of a "proxy user" 812 associated with the schema owner assigned to the specific tenant. This schema owner can access only the tenancy that the schema was created for, and the value of the tenancy is the value of the schema owner. When a request is made for data in database 806, microservice 804 uses the connections in connection pool 810 to provide that data. Accordingly, multi-tenancy is achieved by having stateless, elastic middle tier services process incoming requests in the context of (e.g., in association with) the tenant-specific data store binding established on a per request basis on top of the data connection created in the context of (e.g., in association with) the data store proxy user associated with the resource tenancy, and the database can scale independently of the services.

[00121] The following provides an example functionality for implementing proxy user 812:

```
dbOperation = <prepare DB command to execute>
dbConnection = getDBConnectionFromPool()
dbConnection.setProxyUser (resourceTenant)
result = dbConnection.executeOperation (dbOperation)
```

In this functionality, microservice 804 sets the “Proxy User” setting on the connection pulled from connection pool 810 to the “Tenant,” and performs the database operation in the context of the tenant while using the database connection in connection pool 810.

[00122] When striping every table to configure different columns in a same database for different tenants, one table may include all tenants’ data mixed together.

In contrast, one embodiment provides a tenant-driven data tier. The embodiment does not stripe the same database for different tenants, but instead provides a different physical database per tenant. For example, multi-tenancy may be implemented by using a pluggable database (e.g., Oracle Database 12c from Oracle Corp.) where each tenant is allocated a separate partition. At the data tier, a resource manager processes the request and then asks for the data source for the request (separate from metadata). The embodiment performs runtime switch to a respective data source/store per request. By isolating each tenant's data from the other tenants, the embodiment provides improved data security.

[00123] In one embodiment, various tokens codify different tenancies. A URL token may identify the tenancy of the application that requests a service. An identity token may codify the identity of a user that is to be authenticated. An access token may identify multiple tenancies. For example, an access token may codify the tenancy that is the target of such access (e.g., an application tenancy) as well as the user tenancy of the user that is given access. A client assertion token may identify a client ID and the client tenancy. A user-assertion token may identify the user and the user tenancy.

[00124] In one embodiment, an identity token includes at least a claim/statement indicating the user tenant name (i.e., where the user lives). A "claim" (as used by one of ordinary skill in the security field) in connection with authorization tokens is a statement that one subject makes about itself or another subject. The statement can be about a name, identity, key, group, privilege, or capability, for example. Claims are issued by a provider, and they are given one or more values and then packaged in security tokens

that are issued by an issuer, commonly known as a security token service ("STS").

[00125] In one embodiment, an access token includes at least a claim/statement indicating the resource tenant name at the time the request for the access token was made (e.g., the customer), a claim indicating the user tenant name, a claim indicating the name of the OAuth client making the request, and a claim indicating the client tenant name. In one embodiment, an access token may be implemented according to the following JSON functionality:

```
{
 …
 " tok_type " : "AT",
 "user_id" : "testuser",
 "user_tenantname" : "<value-of-identity-tenant>"
 "tenant" : "<value-of-resource-tenant>"
 "client_id" : "testclient",
 "client_tenantname": "<value-of-client-tenant>"
…
}
```

[00126] In one embodiment, a client assertion token includes at least a claim indicating the client tenant name, and a claim indicating the name of the OAuth client making the request.

[00127] The tokens and/or multiple tenancies described herein may be implemented in any multi-tenant cloud-based service other than IDCS. For example, the tokens and/or multiple tenancies described herein may be implemented in SaaS or Enterprise Resource Planning ("ERP") services.

[00128] Fig. 9 is a block diagram of a network view 900 of IDCS in one embodiment. Fig. 9 illustrates network interactions that are performed in one embodiment between application "zones" 904. Applications are broken into zones

based on the required level of protection and the implementation of connections to various other systems (e.g., SSL zone, no SSL zone, etc.). Some application zones provide services that require access from the inside of IDCS, while some application zones provide services that require access from the outside of IDCS, and some are open access. Accordingly, a respective level of protection is enforced for each zone.

[00129] In the embodiment of Fig. 9, service to service communication is performed using HTTP requests. In one embodiment, IDCS uses the access tokens described herein not only to provide services but also to secure access to and within IDCS itself. In one embodiment, IDCS microservices are exposed through RESTful interfaces and secured by the tokens described herein.

[00130] In the embodiment of Fig. 9, any one of a variety of applications/services 902 may make HTTP calls to IDCS APIs to use IDCS services. In one embodiment, the HTTP requests of applications/services 902 go through an Oracle Public Cloud Load Balancing External Virtual IP address ("VIP") 906 (or other similar technologies), a public cloud web routing tier 908, and an IDCS Load Balancing Internal VIP appliance 910 (or other similar technologies), to be received by IDCS web routing tier 912. IDCS web routing tier 912 receives the requests coming in from the outside or from the inside of IDCS and routes them across either an IDCS platform services tier 914 or an IDCS infrastructure services tier 916. IDCS platform services tier 914 includes IDCS microservices that are invoked from the outside of IDCS, such as OpenID Connect, OAuth, SAML, SCIM, etc. IDCS infrastructure services tier 916 includes supporting microservices that are invoked from the inside of IDCS to support the functionality of

other IDCS microservices. Examples of IDCS infrastructure microservices are UI, SSO, reports, cache, job scheduler, service manager, functionality for making keys, etc. An IDCS cache tier 926 supports caching functionality for IDCS platform services tier 914 and IDCS infrastructure services tier 916.

[00131] By enforcing security both for outside access to IDCS and within IDCS, customers of IDCS can be provided with outstanding security compliance for the applications they run.

[00132] In the embodiment of Fig. 9, other than the data tier 918 which communicates based on Structured Query Language ("SQL") and the ID store tier 920 that communicates based on LDAP, OAuth protocol is used to protect the communication among IDCS components (e.g., microservices) within IDCS, and the same tokens that are used for securing access from the outside of IDCS are also used for security within IDCS. That is, web routing tier 912 uses the same tokens and protocols for processing the requests it receives regardless of whether a request is received from the outside of IDCS or from the inside of IDCS. Accordingly, IDCS provides a single consistent security model for protecting the entire system, thereby allowing for outstanding security compliance since the fewer security models implemented in a system, the more secure the system is.

[00133] In the IDCS cloud environment, applications communicate by making network calls. The network call may be based on an applicable network protocol such as HTTP, Transmission Control Protocol ("TCP"), User Datagram Protocol ("UDP"), etc. For example, an application "X" may communicate with an application "Y" based on

HTTP by exposing application "Y" as an HTTP Uniform Resource Locator ("URL"). In one embodiment, "Y" is an IDCS microservice that exposes a number of resources each corresponding to a capability. When "X" (e.g., another IDCS microservice) needs to call "Y", it constructs a URL that includes "Y" and the resource/capability that needs to be invoked (e.g., https:/host/Y/resource), and makes a corresponding REST call which goes through web routing tier 912 and gets directed to "Y".

[00134] In one embodiment, a caller outside the IDCS may not need to know where "Y" is, but web routing tier 912 needs to know where application "Y" is running. In one embodiment, IDCS implements discovery functionality (implemented by an API of OAuth service) to determine where each application is running so that there is no need for the availability of static routing information.

[00135] In one embodiment, an enterprise manager ("EM") 922 provides a "single pane of glass" that extends on-premise and cloud-based management to IDCS. In one embodiment, a "Chef" server 924 which is a configuration management tool from Chef Software, Inc., provides configuration management functionality for various IDCS tiers. In one embodiment, a service deployment infrastructure and/or a persistent stored module 928 may send OAuth2 HTTP messages to IDCS web routing tier 912 for tenant lifecycle management operations, public cloud lifecycle management operations, or other operations. In one embodiment, IDCS infrastructure services tier 916 may send ID/password HTTP messages to a public cloud notification service 930 or a public cloud storage service 932.

Cloud Access Control - SSO

[00136] One embodiment supports lightweight cloud standards for implementing a cloud scale SSO service. Examples of lightweight cloud standards are HTTP, REST, and any standard that provides access through a browser (since a web browser is lightweight). On the contrary, SOAP is an example of a heavy cloud standard which requires more management, configuration, and tooling to build a client with. The embodiment uses OpenID Connect semantics for applications to request user authentication against IDCS. The embodiment uses lightweight HTTP cookie-based user session tracking to track user's active sessions at IDCS without statefull server-side session support. The embodiment uses JWT-based identity tokens for applications to use in mapping an authenticated identity back to their own local session. The embodiment supports integration with federated identity management systems, and exposes SAML IDP support for enterprise deployments to request user authentication against IDCS.

[00137] Fig. 10 is a block diagram 1000 of a system architecture view of SSO functionality in IDCS in one embodiment. The embodiment enables client applications to leverage standards-based web protocols to initiate user authentication flows. Applications requiring SSO integration with a cloud system may be located in enterprise data centers, in remote partner data centers, or even operated by a customer on-premise. In one embodiment, different IDCS platform services implement the business of SSO, such as OpenID Connect for processing login/logout requests from connected native applications (i.e., applications utilizing OpenID Connect to integrate with IDCS); SAML IDP service for processing browser-based login/logout requests from connected

applications; SAML SP service for orchestrating user authentication against an external SAML IDP; and an internal IDCS SSO service for orchestrating end user login ceremony including local or federated login flows, and for managing IDCS host session cookie. Generally, HTTP works either with a form or without a form. When it works with a form, the form is seen within a browser. When it works without a form, it functions as a client to server communication. Both OpenID Connect and SAML require the ability to render a form, which may be accomplished by presence of a browser or virtually performed by an application that acts as if there is a browser. In one embodiment, an application client implementing user authentication/SSO through IDCS needs to be registered in IDCS as an OAuth2 client and needs to obtain client identifier and credentials (e.g., ID/password, ID/certificate, etc.).

[00138] The example embodiment of Fig. 10 includes three components/microservices that collectively provide login capabilities, including two platform microservices: OAuth2 1004 and SAML2 1006, and one infrastructure microservice: SSO 1008. In the embodiment of Fig. 10, IDCS provides an "Identity Metasystem" in which SSO services 1008 are provided over different types of applications, such as browser based web or native applications 1010 requiring 3-legged OAuth flow and acting as an OpenID Connect relaying party ("RP," an application that outsources its user authentication function to an IDP), native applications 1011 requiring 2-legged OAuth flow and acting as an OpenID Connect RP, and web applications 1012 acting as a SAML SP.

[00139] Generally, an Identity Metasystem is an interoperable architecture for

digital identity, allowing for employing a collection of digital identities based on multiple underlying technologies, implementations, and providers. LDAP, SAML, and OAuth are examples of different security standards that provide identity capability and can be the basis for building applications, and an Identity Metasystem may be configured to provide a unified security system over such applications. The LDAP security model specifies a specific mechanism for handling identity, and all passes through the system are to be strictly protected. SAML was developed to allow one set of applications securely exchange information with another set of applications that belong to a different organization in a different security domain. Since there is no trust between the two applications, SAML was developed to allow for one application to authenticate another application that does not belong to the same organization. OAuth provides OpenID Connect that is a lightweight protocol for performing web based authentication.

[00140] In the embodiment of Fig. 10, when an OpenID application 1010 connects to an OpenID server in IDCS, its "channels" request SSO service. Similarly, when a SAML application 1012 connects to a SAML server in IDCS, its "channels" also request SSO service. In IDCS, a respective microservice (e.g., an OpenID microservice 1004 and a SAML microservice 1006) will handle each of the applications, and these microservices request SSO capability from SSO microservice 1008. This architecture can be expanded to support any number of other security protocols by adding a microservice for each protocol and then using SSO microservice 1008 for SSO capability. SSO microservice 1008 issues the sessions (i.e., an SSO cookie 1014 is provided) and is the only system in the architecture that has the authority to issue a

session. An IDCS session is realized through the use of SSO cookie 1014 by browser 1002. Browser 1002 also uses a local session cookie 1016 to manage its local session.

[00141] In one embodiment, for example, within a browser, a user may use a first application based on SAML and get logged in, and later use a second application built with a different protocol such as OAuth. The user is provided with SSO on the second application within the same browser. Accordingly, the browser is the state or user agent and maintains the cookies.

[00142] In one embodiment, SSO microservice 1008 provides login ceremony 1018, ID/password recovery 1020, first time login flow 1022, an authentication manager 1024, an HTTP cookie manager 1026, and an event manager 1028. Login ceremony 1018 implements SSO functionality based on customer settings and/or application context, and may be configured according to a local form (i.e., basic Auth), an external SAML IDP, an external OIDC IDP, etc. ID/password recovery 1020 is used to recover a user's ID and/or password. First time login flow 1022 is implemented when a user logs in for the first time (i.e., an SSO session does not yet exist). Authentication manager 1024 issues authentication tokens upon successful authentication. HTTP cookie manager 1026 saves the authentication token in an SSO cookie. Event manager 1028 publishes events related to SSO functionality.

[00143] In one embodiment, interactions between OAuth microservice 1004 and SSO microservice 1008 are based on browser redirects so that SSO microservice 1008 challenges the user using an HTML form, validates credentials, and issues a session cookie.

[00144] In one embodiment, for example, OAuth microservice 1004 may receive an authorization request from browser 1002 to authenticate a user of an application according to 3-legged OAuth flow. OAuth microservice 1004 then acts as an OIDC provider 1030, redirects browser 1002 to SSO microservice 1008, and passes along application context. Depending on whether the user has a valid SSO session or not, SSO microservice 1008 either validates the existing session or performs a login ceremony. Upon successful authentication or validation, SSO microservice 1008 returns authentication context to OAuth microservice 1004. OAuth microservice 1004 then redirects browser 1002 to a callback URL with an authorization ("AZ") code. Browser 1002 sends the AZ code to OAuth microservice 1004 to request the required tokens 1032. Browser 1002 also includes its client credentials (obtained when registering in IDCS as an OAuth2 client) in the HTTP authorization header. OAuth microservice 1004 in return provides the required tokens 1032 to browser 1002. In one embodiment, tokens 1032 provided to browser 1002 include JW identity and access tokens signed by the IDCS OAuth2 server. Further details of this functionality are disclosed below with reference to Fig. 11.

[00145] In one embodiment, for example, OAuth microservice 1004 may receive an authorization request from a native application 1011 to authenticate a user according to a 2-legged OAuth flow. In this case, an authentication manager 1034 in OAuth microservice 1004 performs the corresponding authentication (e.g., based on ID/password received from a client 1011) and a token manager 1036 issues a corresponding access token upon successful authentication.

[00146] In one embodiment, for example, SAML microservice 1006 may receive an SSO POST request from a browser to authenticate a user of a web application 1012 that acts as a SAML SP. SAML microservice 1006 then acts as a SAML IDP 1038, redirects browser 1002 to SSO microservice 1008, and passes along application context. Depending on whether the user has a valid SSO session or not, SSO microservice 1008 either validates the existing session or performs a login ceremony. Upon successful authentication or validation, SSO microservice 1008 returns authentication context to SAML microservice 1006. SAML microservice then redirects to the SP with required tokens.

[00147] In one embodiment, for example, SAML microservice 1006 may act as a SAML SP 1040 and go to a remote SAML IDP 1042 (e.g., an active directory federation service ("ADFS")). One embodiment implements the standard SAML/AD flows. In one embodiment, interactions between SAML microservice 1006 and SSO microservice 1008 are based on browser redirects so that SSO microservice 1008 challenges the user using an HTML form, validates credentials, and issues a session cookie.

[00148] In one embodiment, the interactions between a component within IDCS (e.g., 1004, 1006, 1008) and a component outside IDCS (e.g., 1002, 1011, 1042) are performed through firewalls 1044.

Login/Logout Flow

[00149] Fig. 11 is a message sequence flow 1100 of SSO functionality provided by IDCS in one embodiment. When a user uses a browser 1102 to access a client 1106 (e.g., a browser-based application or a mobile/native application), Cloud Gate

1104 acts as an application enforcement point and enforces a policy defined in a local policy text file. If Cloud Gate 1104 detects that the user has no local application session, it requires the user to be authenticated. In order to do so, Cloud Gate 1104 redirects browser 1102 to OAuth2 microservice 1110 to initiate OpenID Connect login flow against the OAuth2 microservice 1110 (3-legged AZ Grant flow with scopes = "openid profile").

[00150] The request of browser 1102 traverses IDCS routing tier web service 1108 and Cloud Gate 1104 and reaches OAuth2 microservice 1110. OAuth2 microservice 1110 constructs the application context (i.e., metadata that describes the application, e.g., identity of the connecting application, client ID, configuration, what the application can do, etc.), and redirects browser 1102 to SSO microservice 1112 to log in.

[00151] If the user has a valid SSO session, SSO microservice 1112 validates the existing session without starting a login ceremony. If the user does not have a valid SSO session (i.e., no session cookie exists), the SSO microservice 1112 initiates the user login ceremony in accordance with customer's login preferences (e.g., displaying a branded login page). In order to do so, the SSO microservice 1112 redirects browser 1102 to a login application service 1114 implemented in JavaScript. Login application service 1114 provides a login page in browser 1102. Browser 1102 sends a REST POST to the SSO microservice 1112 including login credentials. The SSO microservice 1112 generates an access token and sends it to Cloud Gate 1104 in a REST POST. Cloud Gate 1104 sends the authentication information to Admin SCIM microservice

1116 to validate the user's password. Admin SCIM microservice 1116 determines successful authentication and sends a corresponding message to SSO microservice 1112.

[00152] In one embodiment, during the login ceremony, the login page does not display a consent page, as "login" operation requires no further consent. Instead, a privacy policy is stated on the login page, informing the user about certain profile attributes being exposed to applications. During the login ceremony, the SSO microservice 1112 respects customer's IDP preferences, and if configured, redirects to the IDP for authentication against the configured IDP.

[00153] Upon successful authentication or validation, SSO microservice 1112 redirects browser 1102 back to OAuth2 microservice 1110 with the newly created/updated SSO host HTTP cookie (e.g., the cookie that is created in the context of the host indicated by "HOSTURL") containing the user's authentication token. OAuth2 microservice 1110 returns AZ Code (e.g., an OAuth concept) back to browser 1102 and redirects to Cloud Gate 1104. Browser 1102 sends AZ Code to Cloud Gate 1104, and Cloud Gate 1104 sends a REST POST to OAuth2 microservice 1110 to request the access token and the identity token. Both tokens are scoped to OAuth microservice 1110 (indicated by the audience token claim). Cloud Gate 1104 receives the tokens from OAuth2 microservice 1110.

[00154] Cloud Gate 1104 uses the identity token to map the user's authenticated identity to its internal account representation, and it may save this mapping in its own HTTP cookie. Cloud Gate 1104 then redirects browser 1102 to client 1106. Browser

1102 then reaches client 1106 and receives a corresponding response from client 1106. From this point on, browser 1102 can access the application (i.e., client 1106) seamlessly for as long as the application's local cookie is valid. Once the local cookie becomes invalid, the authentication process is repeated.

[00155] Cloud Gate 1104 further uses the access token received in a request to obtain "userinfo" from OAuth2 microservice 1110 or the SCIM microservice. The access token is sufficient to access the "userinfo" resource for the attributes allowed by the "profile" scope. It is also sufficient to access "/me" resources via the SCIM microservice. In one embodiment, by default, the received access token is only good for user profile attributes that are allowed under the "profile" scope. Access to other profile attributes is authorized based on additional (optional) scopes submitted in the AZ grant login request issued by Cloud Gate 1104.

[00156] When the user accesses another OAuth2 integrated connecting application, the same process repeats.

[00157] In one embodiment, the SSO integration architecture uses a similar OpenID Connect user authentication flow for browser-based user logouts. In one embodiment, a user with an existing application session accesses Cloud Gate 1104 to initiate a logout. Alternatively, the user may have initiated the logout on the IDCS side. Cloud Gate 1104 terminates the application-specific user session, and initiates OAuth2 OpenID Provider ("OP") logout request against OAuth2 microservice 1110. OAuth2 microservice 1110 redirects to SSO microservice 1112 that kills the user's host SSO cookie. SSO microservice 1112 initiates a set of redirects (OAuth2 OP and SAML IDP)

against known logout endpoints as tracked in user's SSO cookie.

[00158] In one embodiment, if Cloud Gate 1104 uses SAML protocol to request user authentication (e.g., login), a similar process starts between the SAML microservice and SSO microservice 1112.

Cloud Cache

[00159] One embodiment provides a service/capability referred to as Cloud Cache. Cloud Cache is provided in IDCS to support communication with applications that are LDAP based (e.g., email servers, calendar servers, some business applications, etc.) since IDCS does not communicate according to LDAP while such applications are configured to communicate only based on LDAP. Typically, cloud directories are exposed via REST APIs and do not communicate according to the LDAP protocol. Generally, managing LDAP connections across corporate firewalls requires special configurations that are difficult to set up and manage.

[00160] To support LDAP based applications, Cloud Cache translates LDAP communications to a protocol suitable for communication with a cloud system. Generally, an LDAP based application uses a database via LDAP. An application may be alternatively configured to use a database via a different protocol such as SQL. However, LDAP provides a hierarchical representation of resources in tree structures, while SQL represents data as tables and fields. Accordingly, LDAP may be more desirable for searching functionality, while SQL may be more desirable for transactional functionality.

[00161] In one embodiment, services provided by IDCS may be used in an LDAP

based application to, for example, authenticate a user of the applications (i.e., an identity service) or enforce a security policy for the application (i.e., a security service). In one embodiment, the interface with IDCS is through a firewall and based on HTTP (e.g., REST). Typically, corporate firewalls do not allow access to internal LDAP communication even if the communication implements Secure Sockets Layer ("SSL"), and do not allow a TCP port to be exposed through the firewall. However, Cloud Cache translates between LDAP and HTTP to allow LDAP based applications reach services provided by IDCS, and the firewall will be open for HTTP.

[00162] Generally, an LDAP directory may be used in a line of business such as marketing and development, and defines users, groups, works, etc. In one example, a marketing and development business may have different targeted customers, and for each customer, may have their own applications, users, groups, works, etc. Another example of a line of business that may run an LDAP cache directory is a wireless service provider. In this case, each call made by a user of the wireless service provider authenticates the user's device against the LDAP directory, and some of the corresponding information in the LDAP directory may be synchronized with a billing system. In these examples, LDAP provides functionality to physically segregate content that is being searched at runtime.

[00163] In one example, a wireless service provider may handle its own identity management services for their core business (e.g., regular calls), while using services provided by IDCS in support of a short term marketing campaign. In this case, Cloud Cache "flattens" LDAP when it has a single set of users and a single set of groups that it

runs against the cloud. In one embodiment, any number of Cloud Caches may be implemented in IDCS.

Distributed Data Grid

[00164] In one embodiment, the cache cluster in IDCS is implemented based on a distributed data grid, as disclosed, for example, in U.S. Pat. Pub. No. 2016/0092540, the disclosure of which is hereby incorporated by reference. A distributed data grid is a system in which a collection of computer servers work together in one or more clusters to manage information and related operations, such as computations, within a distributed or clustered environment. A distributed data grid can be used to manage application objects and data that are shared across the servers. A distributed data grid provides low response time, high throughput, predictable scalability, continuous availability, and information reliability. In particular examples, distributed data grids, such as, e.g., the Oracle Coherence data grid from Oracle Corp., store information in-memory to achieve higher performance, and employ redundancy in keeping copies of that information synchronized across multiple servers, thus ensuring resiliency of the system and continued availability of the data in the event of failure of a server.

[00165] In one embodiment, IDCS implements a distributed data grid such as Coherence so that every microservice can request access to shared cache objects without getting blocked. Coherence is a proprietary Java-based in-memory data grid, designed to have better reliability, scalability, and performance than traditional relational database management systems. Coherence provides a peer to peer (i.e., with no central manager), in-memory, distributed cache.

[00166] Fig. 12 illustrates an example of a distributed data grid 1200 which stores data and provides data access to clients 1250 and implements embodiments of the invention. A "data grid cluster", or "distributed data grid", is a system comprising a plurality of computer servers (e.g., 1220a, 1220b, 1220c, and 1220d) which work together in one or more clusters (e.g., 1200a, 1200b, 1200c) to store and manage information and related operations, such as computations, within a distributed or clustered environment. While distributed data grid 1200 is illustrated as comprising four servers 1220a, 1220b, 1220c, 1220d, with five data nodes 1230a, 1230b, 1230c, 1230d, and 1230e in a cluster 1200a, the distributed data grid 1200 may comprise any number of clusters and any number of servers and/or nodes in each cluster. In an embodiment, distributed data grid 1200 implements the present invention.

[00167] As illustrated in Fig. 12, a distributed data grid provides data storage and management capabilities by distributing data over a number of servers (e.g., 1220a, 1220b, 1220c, and 1220d) working together. Each server of the data grid cluster may be a conventional computer system such as, for example, a "commodity x86" server hardware platform with one to two processor sockets and two to four CPU cores per processor socket. Each server (e.g., 1220a, 1220b, 1220c, and 1220d) is configured with one or more CPUs, Network Interface Cards (“NIC”), and memory including, for example, a minimum of 4 GB of RAM up to 64 GB of RAM or more. Server 1220a is illustrated as having CPU 1222a, Memory 1224a, and NIC 1226a (these elements are also present but not shown in the other Servers 1220b, 1220c, 1220d). Optionally, each server may also be provided with flash memory (e.g., SSD 1228a) to provide spillover

storage capacity. When provided, the SSD capacity is preferably ten times the size of the RAM. The servers (e.g., 1220a, 1220b, 1220c, 1220d) in a data grid cluster 1200a are connected using high bandwidth NICs (e.g., PCI-X or PCIe) to a high-performance network switch 1220 (for example, gigabit Ethernet or better).

[00168] A cluster 1200a preferably contains a minimum of four physical servers to avoid the possibility of data loss during a failure, but a typical installation has many more servers. Failover and failback are more efficient the more servers that are present in each cluster and the impact of a server failure on a cluster is lessened. To minimize communication time between servers, each data grid cluster is ideally confined to a single switch 1202 which provides single hop communication between servers. A cluster may thus be limited by the number of ports on the switch 1202. A typical cluster will therefore include between 4 and 96 physical servers.

[00169] In most Wide Area Network ("WAN") configurations of a distributed data grid 1200, each data center in the WAN has independent, but interconnected, data grid clusters (e.g., 1200a, 1200b, and 1200c). A WAN may, for example, include many more clusters than shown in Fig. 12. Additionally, by using interconnected but independent clusters (e.g., 1200a, 1200b, 1200c) and/or locating interconnected, but independent, clusters in data centers that are remote from one another, the distributed data grid can secure data and service to clients 1250 against simultaneous loss of all servers in one cluster caused by a natural disaster, fire, flooding, extended power loss, and the like.

[00170] One or more nodes (e.g., 1230a, 1230b, 1230c, 1230d and 1230e)

operate on each server (e.g., 1220a, 1220b, 1220c, 1220d) of a cluster 1200a. In a distributed data grid, the nodes may be, for example, software applications, virtual machines, or the like, and the servers may comprise an operating system, hypervisor, or the like (not shown) on which the node operates. In an Oracle Coherence data grid, each node is a Java virtual machine ("JVM"). A number of JVMs/nodes may be provided on each server depending on the CPU processing power and memory available on the server. JVMs/nodes may be added, started, stopped, and deleted as required by the distributed data grid. JVMs that run Oracle Coherence automatically join and cluster when started. JVMs/nodes that join a cluster are called cluster members or cluster nodes.

[00171] Each client or server includes a bus or other communication mechanism for communicating information, and a processor coupled to bus for processing information. The processor may be any type of general or specific purpose processor. Each client or server may further include a memory for storing information and instructions to be executed by processor. The memory can be comprised of any combination of random access memory ("RAM"), read only memory ("ROM"), static storage such as a magnetic or optical disk, or any other type of computer readable media. Each client or server may further include a communication device, such as a network interface card, to provide access to a network. Therefore, a user may interface with each client or server directly, or remotely through a network, or any other method.

[00172] Computer readable media may be any available media that can be accessed by processor and includes both volatile and non-volatile media, removable

and non-removable media, and communication media. Communication media may include computer readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism, and includes any information delivery media.

[00173] The processor may further be coupled via bus to a display, such as a Liquid Crystal Display ("LCD"). A keyboard and a cursor control device, such as a computer mouse, may be further coupled to bus to enable a user to interface with each client or server.

[00174] In one embodiment, the memory stores software modules that provide functionality when executed by the processor. The modules include an operating system that provides operating system functionality each client or server. The modules may further include a cloud identity management module for providing cloud identity management functionality, and all other functionality disclosed herein.

[00175] The clients may access a web service such as a cloud service. The web service may be implemented on a WebLogic Server from Oracle Corp. in one embodiment. In other embodiments, other implementations of a web service can be used. The web service accesses a database which stores cloud data.

Data Management – Metadata Driven Framework

[00176] In embodiments, there is a need to manage a large amount of different types of data/resources, referred to as "resource types". Examples of resource types include users, groups, applications ("apps"), configurations, settings, password policies, etc. The resource types are management entities that are managed in the cloud using

embodiments of the invention.

[00177] Each resource type has various configurations that define its behavior. One configuration is a schema (i.e., data that can be associated with a resource of that type), including both primary/core and extension schemas. For example, if a resource type is a "user", each user may have 4-5 schemas associated with it. A core schema includes the user name (e.g., first, last), phone number, address, and any relevant profile information for that user. Extension schemas may include a password state (e.g., has the user's password expired, has the maximum number of password tries been exceeded. etc.) or a user state (e.g., active or inactive). These are considered "schema attributes". Similarly, another resource type such as application has its own schema that defines how the resource is organized and how its data is related..

[00178] Every schema has a list of attributes in one embodiment. Every attribute has a set of metadata that defines the behavior of that attribute (e.g., is it read/write, what is its mutability (i.e., ability to be changed or altered), is it required or not, is it searchable). For each resource type, there can be a resource type definition and a schema definition. A resource type definition can include: (1) a list of schemas - core and optional extension schemas; (2) Supported operations - create, replace, update, delete, get, search, postsearch; (3) Data provider type: LDAP provider, database ("DB") provider, Notification provider, etc. Schema definitions can include a list of attributes and the metadata for each of the attributes describing its properties and behavior - data type, mutability (readonly, readwrite, immutable, writeonly), returned (always, request, never), target attribute name, trimspaces, maxlength, etc. Therefore, the metadata

makes up the definition of the resource type, what schemas are associated with it, and for each attribute what the metadata is.

[00179] In one example embodiment, there are 175 different resource types. If software code was required to manage each one of those resource types and each specific schema, there would be a requirement for a large amount of code specific to each resource type and schema. For example, for a "POJO" (plain old Java object) resource type that provides getters and setters for a particular attribute (e.g., get user name, set user name, get first name, get last name), it would require a large amount of code for all resource types.

[00180] In contrast, in embodiments of the invention, a single unified code/module performs all functionality (e.g., create, update, modify, delete, search, etc.) for any resource type. Initially, embodiments determine what the resource type is, and then look up the schema and schema definitions to determine the attributes and the attribute definitions. Embodiments can then make runtime decisions such as which attributes are required for this resource type for its schemas in order to perform validation. When performing the persistence of the data in the database in LDAP, embodiments are aware of what the attributes are for the resource type so embodiments can construct the payload correctly for SQL update/insert. For each resource, resource type and schema definitions contain the provider type and target attribute mapping configured. If the provider type is LDAP then an LDAP provider is invoked to persist the data in LDAP with the right objectclass under the directory tree as configured in the resource type and schema definition. If the provider type is a DB

provider then the provider generates SQLs to persist/fetch data in the DB in a table in right columns as configured in resource type and schema definitions.

[00181] Therefore, one embodiment is entirely data driven so that if a developer wants to add a new resource type, the developer merely needs to add the JSON definitions of the new resource type and schema instead of writing new code. Embodiments are considered a data driven model since embodiments are completely metadata driven. Embodiments perform a complete abstraction of the data away from the run-time code. As a result, embodiments do not require the need to write, maintain or test separate code for each resource type.

[00182] In one embodiment, for performance reasons, all metadata is cached at service startup time. Further, embodiments can use metadata to drive documentation and can automatically generate external docs, including REST APIs, based on resource types and schema definitions.

[00183] Fig. 13 illustrates a data manager architecture for IDCS or Identity as a Service ("IDaaS") in accordance with one embodiment. The architecture includes a resource manager 1301 (or "ResourceManager"), a SCIM/REST layer 1310, an API layer 1311, a data provider ("DP") layer 1312 and a data store layer 1313.

[00184] Resource manager 1301 is the common data access layer for IDCS/IDaaS. It is metadata driven so it can manage any resource type defined in a SCIM compliant resource type and schema definition. In one embodiment, resource manager 1301 handles all common logic for all resource types including schema based validation of attributes, data types, required, canonical values, etc. It also handles

setting default values, create/update dates, create/update by, protecting sensitive attributes, mapping to target attributes, authorization checks and event publishing. Resource type specific managers in API layer 1311 extend resource manager 1301 to handle any resource type specific operations or extend common logic for CRUDQ operations. Resource manager 1301 integrates with data stores in data store layer 1313 via data providers in data provider layer 1313, which integrate directly with the data store specific interfaces. Since resource manager 1301 is metadata driven, it supports runtime schema customizations and DataProvider configuration changes with no impact to the resource manager 1301 or resource type managers 1311.

[00185] A resource type (or "ResourceType") is a type of resource managed by IDCS. Examples include User, Group, Application, Token, Key, etc. In SCIM, each resource type is a top-level endpoint (e.g. /Users, /Groups). Each resource type has a resource type definition (or "ResourceTypeDef"). A resource type definition is the metadata that describes a given resource type. A resource type definition defines the resource type name, endpoint, primary schema URI and extended schema URIs, if any. In addition it defines several IDCS specific metadata. Each resource type definition is a blob of JSON. Resource types can be pre-seeded or can be created at runtime. Any resource type definition can be configured to be internal, meaning it is not accessible via SCIM REST 1310, or external, meaning it is discoverable through SCIM REST GET /ResourceTypes and GET /Schemas/<ResourceType>.

[00186] A schema definition (or "SchemaDef") is a collection of attribute definitions (or "AttributeDefs") that describe the contents of an entire or partial resource

type (e.g., “Device”). SCIM defines the core metadata and IDCS, in accordance to embodiments, extends the metadata. Specifically, a schema definition describes every attribute and sub-attribute of that resource type. Each schema definition is a blob of JSON. Schema definitions can be pre-seeded or can be created at runtime.

[00187] An attribute definition defines the name and metadata such as type (e.g., string, binary), cardinality (single, multi, complex), mutability (readOnly, readWrite, etc.), returnability, searchability, etc. Attribute names should be camel-cased (e.g., camelCased) and should be unique within the schema they are defined in. Attribute data types can be one of the following in an embodiment: String; Boolean; Decimal; Integer; DateTime; Binary; Reference; or Complex.

[00188] A singular attribute is a resource attribute that contains 0..1 values (e.g., “displayName”). A multi-valued attribute is a resource attribute that contains 0..n values (e.g., “emails”). A simple attribute is a singular or multi-valued attribute whose value is a primitive (e.g., “String”). A complex attribute is a singular or multi-valued attribute whose value is a composition of one or more simple attributes (e.g., “addresses”), and has sub-attributes (e.g., “streetAddress”, “locality”, “postalCode” and “country”). A sub-attribute is a simple attribute that is contained within a complex attribute.

[00189] A resource is an instance of an IDCS managed artifact containing one or more attributes. In SCIM, a resource is an object that can be read and manipulated, such as a particular User, Group or Token. Each resource has a globally unique identifier and contains attribute values that conform to the corresponding resource type definition’s schemas.

[00190] In one embodiment, in API layer 1311, a Java class is implemented for each resource type as a resource type manager. A user manager, for example, manages users. A group manager manages groups. Each manager exposes the interfaces appropriate for managing objects of that resource type. Every resource type manager extends a common abstract resource manager that implements methods to create, replace, update, delete, get and search for resources. Each resource type manager can implement custom validation for each method implemented by abstract resource manager, if needed. In addition, each resource type manager can extend these methods as needed. For example, the user manager exposes methods unique to the user resource including enable, disable, lock, unlock, changePassword. The group manager exposes methods that the user manager does not, such as grant or revoke user membership.

[00191] In one embodiment, HK2 based on the JSR-330 standard annotations will be used for API layer 1311. Each resource type manager's custom Java interface will be annotated with @Contract and their impls will be annotated with @Service. This will ensure that that the resource type manager class will be placed into the service registry advertised under both resource type manager and <ResourceType>ManagerImpl and will be requestable via a service locator.

[00192] In one embodiment, resource manager 1301 is a stateless common Java class, which defines a set of APIs for querying and managing resources of any resource type. Resource manager's interfaces will be annotated with @Contract and <ResourceType>Manager's impls will be annotated with @Service to ensure they will

be placed into the service registry and requestable via ServiceLocator.

[00193] An abstract resource manager (or "AbstractResourceManager") implements resource manager interfaces providing common behavior that each <ResourceType>Manager inherits. For example, AbstractResourceManager checks authorization, performs validation based on the ResourceTypeDef and emits events for create, replace, update and delete operations. In addition, AbstractResourceManager methods will call back to <ResourceType>Manager's to enable custom validation.

[00194] Data provider layer 1312 is a pluggable layer underneath resource manager 1301. It implements each operation against the underlying data store. For example, the JDBC data provider uses JDBC to talk to the database. The JNDI data provider uses JNDI to talk to a directory service. The data provider will switch between data stores based on the tenant ID of each request.

[00195] One embodiment initially supports two types of data stores: JDBC and JNDI. Other embodiments support other data stores, such as NoSQL. Data stores are tenant specific and can be resource type specific. For example, events may be stored in a JDBC database separate from the JDBC database used to store applications.

[00196] As described, embodiments are data driven because common resource manager 1301 not only handles the validation, create, and error handling and exceptions, regardless of the resource type, but also generates events based on the metadata. The events are queued in a messaging service and are handled by backend handlers and are audited, generate notifications, etc.

[00197] Fig. 14 illustrates the functional flow of embodiments of the invention

implemented by the resource data manager of Fig. 13. The functionality includes:

(1) At 1401, resolve what the resource type is.

(2) At 1402, validate that the operation is supported by the resource type (i.e., validate what the user is trying to do). For example, some resource types support create, update, and delete, and some only support get and search. The support is determined by the metadata.

(3) At 1403, authorize by making an authorization check to determine if authorized to perform the operation.

(4) At 1404, 1405, callbacks to the resource manager are implemented if required (i.e., certain resource types require customization) for, for example, custom validation, custom preprocessing, custom post-processing, custom event generation. The callbacks are invoked from the common flow, and is novel because it allows each resource type to inject resource type specific behavior.

(5) At 1406, get the data provider based on the request. For a given resource type there is only one data provider irrespective of the tenant. Based on the tenant, the data provider establishes a connection to the correct database schema (for resources stored in DB) or points to the right directory tree (for resources stored in LDAP). Uses a “Getdataprovider” Java method in one embodiment.

(6) At 1406, call the data provider to perform the operation (e.g., in the example of Fig. 14, the operation is “create”).

(7) At 1407, after the create operation, make callback to resource manager to determine if any post processing is needed (e.g., inject additional attributes in result, modify data). Provides a plug-in callback to the resource manager.

(8) At 1408, publish events.

(9) At 1409, return resource that comes back in a POST response.

[00198] The primary use of metadata in the flow in Fig. 14 in one embodiment is validation of payload that comes in with the Post request 1409 against the schema defined for that resource type. This is dynamically done based on the cached resource data. However, resource type metadata is not only for validation, as the metadata that is loaded at the beginning of the request can be used at every stage in resource manager processing, including: Validation 1402 (check for valid attrnames, data type, missing required attrs, etc.), Authorization 1407, Dataprovider operation 1430 (target attribute mapping, Table or objectclass mapping, etc.), Post processing 1409 (filter out the data based on return attribute property in metadata), Publish events 1408 (what events to publish is in metadata).

[00199] Embodiments determine where the requests are coming from, and who is the user of the resource manager. The user can be an end user, another application, an internal IDCS component, etc. Typically, a client makes the request - an API request in the form of an HTTP request that is then handed over to the resource manager to process based on the resource metadata.

[00200] In one embodiment, resource manager 1301 is a microservice that is referred to as “admin service” and handles the administration of all resource types. It talks to a messaging service (i.e., a microservice) when it generates events. Referring again to Fig. 13, resource manager 1301 represents the admin service and the entire Fig. 13 (with the exception of the databases at the bottom) is the admin service microservice.

[00201] One embodiment supports the cache coherence of IDCS. For example, for Get request: get/user/ID, embodiments will initially query the cache data provider to see if the User ID is cached. If cached, the data is returned from the cache. If not cached, the data is retrieved from a data store, and the data is added to the coherence cache on the way back.

[00202] Based on the tenant, embodiments determine which DB (e.g., LDAP data partition "DP" 1305 of Fig. 13) to read from using the metadata. Embodiments perform a DB switch at the data layer based on the tenant.

[00203] Embodiments can be used by any service that needs to manage data. The clients that use embodiments can include a UI service console, import jobs, or anything that is updating data in IDCS.

Data Management – Multi-Tenant

[00204] One embodiment implements multi-tenant support at the data layer. Resource manager 1301 processes the request, then asks for the appropriate data source for the request. The processing is tenant driven and separate from the metadata functionality. This functionality provides embodiments with the ability to make runtime switches between data stores based on the tenant, which helps in security (i.e., isolation of tenant data).

[00205] Other known identity managers may not have multiple tenants. Instead, some known identity manager systems perform striping using a different column in the database for every table for each tenant. In these solutions, one table would have all of the multiple tenants data mixed together, which may not be secure.

[00206] In contrast, in one embodiment different databases are used per tenant instead of striping. Embodiments perform a runtime switch to the appropriate data source per request.

Data Management – Automatic Schema Versioning

[00207] In one embodiment, over the life of a resource type (e.g., a "user" resource type), a version 1 may have a schema for a user. In a subsequent version 2 of the resource type, there may be a need to add or delete attributes, so there may be a need to replicate the schema that represents a schema with all attributes of version 1, and a version 2 schema with all attributes of version 2. There may be a need to keep replicating schemas with each new version.

[00208] However, in contrast, in one embodiment, instead of replicating schemas, the schema attributes themselves are allowed to be tagged with either an added since version attribute or a deprecated since version attribute.

[00209] For example, an embodiment may have a version 1 attribute for a resource type called "red". In version 2, red is no longer needed. In the single user schema, embodiments will tag the red attribute as deprecated for version 2, but can also add 3 new attributes to version 2: A, B and C. For the three new attributes, a tag is added for version 2.

[00210] At runtime, when a user makes a request, the request can include the version of the schema they wish to work with (e.g., version 1 users, version 2 users, etc.). At runtime, embodiments evaluate the schema (metadata driven) to determine what the schema version 1 includes, what the schema version 2 includes, etc.

[00211] In an example use case: request to Get a user for version 2 of schema. The request gets the user based on attributes that were added and deprecated. So instead of returning deprecated attributes, the request will return added attributes. In contrast, version 1 would not have any tags added or deprecated.

[00212] Embodiments are entirely metadata driven. This allows multiple versions of every user being supported concurrently through same the resource manager service. Embodiments can support schema changes with zero down time.

[00213] Fig. 15 illustrates automatic schema versioning in accordance with one embodiment. As shown in Fig. 15, version 1 of the user schema includes attributes "name" and "type" (at 1501). Version 2 includes "name" and "costcenter" (at 1502). In this example, the attribute "Type" was deprecated since it is not included in version 2.

[00214] When a request is made to get a user, by default the latest version of the schema (i.e., version 2 in Fig. 15) is always retrieved in one embodiment. However, the request payload can request version 1 instead. In this case, embodiments will get a subset of data specific to that schema. Embodiments also works with cache.

[00215] Most known solutions have a separate schema definition for each version. In contrast, embodiments only have a single schema definition, and metadata that defines the changes between versions. One benefit is the support of zero down time.

[00216] As disclosed, embodiments implement metadata that defines a resource type and associated schemas. A request for performing an operation on a resource in a multi-tenant system is resolved using the metadata to determine the data provider

associated with the tenant that performs the operation.

[00217] Several embodiments are specifically illustrated and/or described herein. However, it will be appreciated that modifications and variations of the disclosed embodiments are covered by the above teachings and within the purview of the appended claims without departing from the spirit and intended scope of the invention.

WHAT IS CLAIMED IS:

1. A non-transitory computer readable medium having instructions stored thereon that, when executed by a processor, cause the processor to provide cloud based identity management, the providing comprising:

receiving a request from an application by a web gate for a resource, wherein the request comprises an operation on a resource type out of a plurality of resource types and the request specifies a tenant out of a plurality of tenants;

accessing a microservice based on the request;

resolving the resource type;

validating that the operation is supported by the resource type based on metadata;

getting a data provider associated with the tenant;

calling the data provider to perform the operation; and

returning the resource.

2. The non-transitory computer readable medium of claim 1, wherein the resource type comprises a schema definition, and the schema definition comprises a plurality of attributes and metadata for each of the attributes.

3. The non-transitory computer readable medium of claim 1, wherein the resolving the resource type comprises determining the resource type and retrieving corresponding schema and schema definitions.

4. The non-transitory computer readable medium of claim 3, the resolving further comprising determining which attributes are required for the resource type based on the schema in order to perform the validating.

5. The non-transitory computer readable medium of claim 1, wherein for multiple versions of the resource type, at least one version of the resource type comprises a tag indicating a deprecated attribute with respect to a previous version, and at least one version of the resource type comprises a tag indicating an added attribute with respect to a previous resource type, further comprising:

performing the operation using the version of the resource type based on corresponding tags of the resource type.

6. The non-transitory computer readable medium of claim 1, wherein the operation comprises one of create, update, delete, get or search.

7. The non-transitory computer readable medium of claim 3, wherein the resource type is a user, and the corresponding schema comprises a password state.

8. The non-transitory computer readable medium of claim 1, wherein the data provider comprises one of a database or an Lightweight Directory Access Protocol (LDAP) provider.

9. A method of providing cloud based identity management, the method comprising:

receiving a request from an application by a web gate for a resource, wherein the request comprises an operation on a resource type out of a plurality of resource types and the request specifies a tenant out of a plurality of tenants;

accessing a microservice based on the request;

resolving the resource type;

validating that the operation is supported by the resource type based on metadata;

getting a data provider associated with the tenant;

calling the data provider to perform the operation; and

returning the resource.

10. The method of claim 9, wherein the resource type comprises a schema definition, and the schema definition comprises a plurality of attributes and metadata for each of the attributes.

11. The method of claim 9, wherein the resolving the resource type comprises determining the resource type and retrieving corresponding schema and schema definitions.

12. The method of claim 11, the resolving further comprising determining which

attributes are required for the resource type based on the schema in order to perform the validating.

13. The method of claim 9, wherein for multiple versions of the resource type, at least one version of the resource type comprises a tag indicating a deprecated attribute with respect to a previous version, and at least one version of the resource type comprises a tag indicating an added attribute with respect to a previous resource type, further comprising:

performing the operation using the version of the resource type based on corresponding tags of the resource type.

14. The method of claim 9, wherein the operation comprises one of create, update, delete, get or search.

15. The method of claim 12, wherein the resource type is a user, and the corresponding schema comprises a password state.

16. The method of claim 9, wherein the data provider comprises one of a database or an Lightweight Directory Access Protocol (LDAP) provider.

17. A system for providing cloud based identity and access management, comprising:

a plurality of tenants;

a plurality of microservices; and

one or more processors that:

receive a request from an application by a web gate for a resource, wherein the request comprises an operation on a resource type out of a plurality of resource types and the request specifies a tenant out of the plurality of tenants;

access a microservice out of the plurality of microservices based on the request;

resolve the resource type;

validate that the operation is supported by the resource type based on metadata;

get a data provider associated with the tenant;

call the data provider to perform the operation; and

return the resource.

18. The system of claim 17, wherein the resource type comprises a schema definition, and the schema definition comprises a plurality of attributes and metadata for each of the attributes.

19. The system of claim 17, wherein the resolve the resource type comprises determining the resource type and retrieving corresponding schema and schema

definitions.

20. The system of claim 17, the resolve further comprising determining which attributes are required for the resource type based on the schema in order to perform the validate.

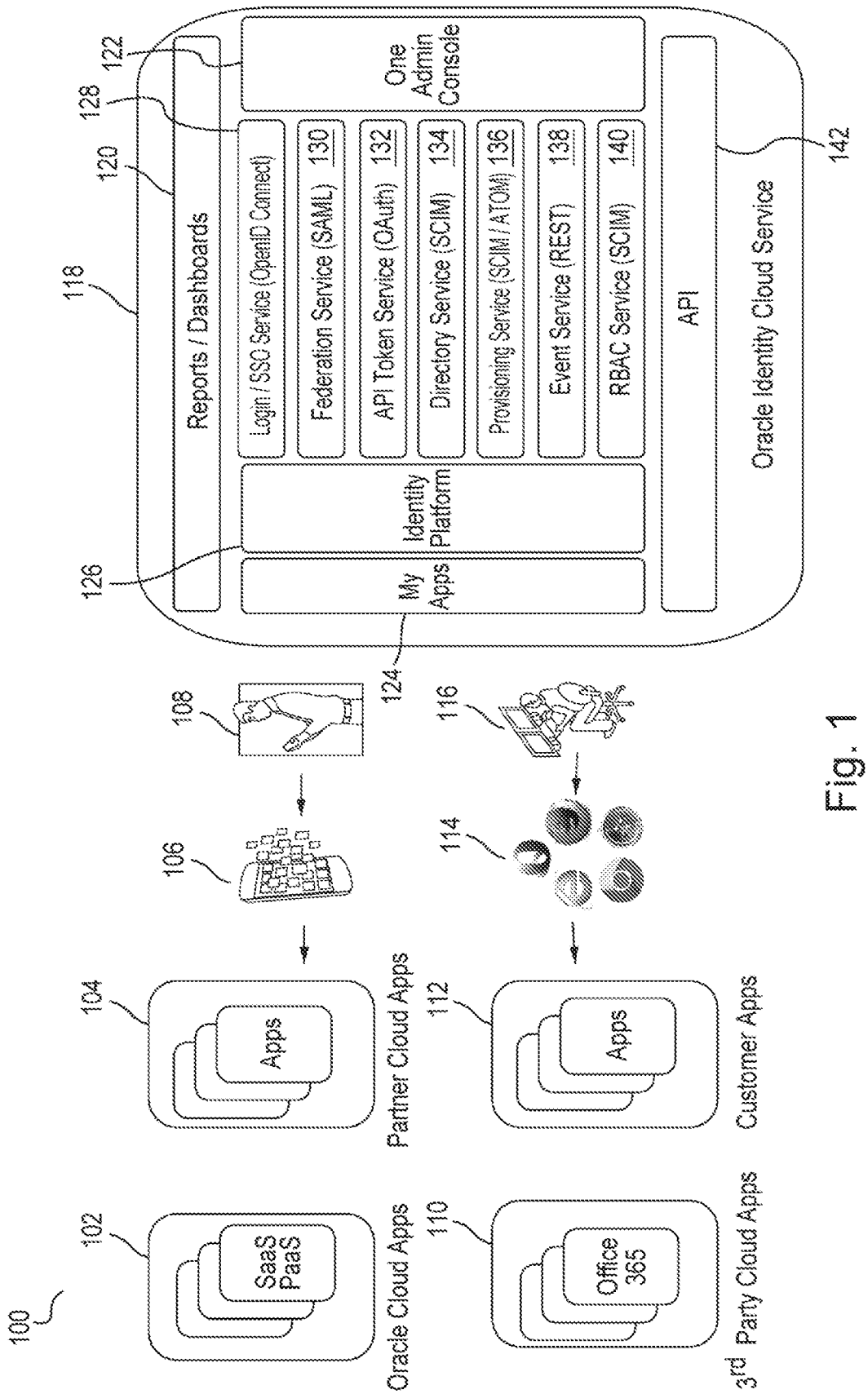
100
102
SaaS
PaaS
Oracle Cloud Apps
104
Apps
Partner Cloud Apps
106
108
110
Office 365
3rd Party Cloud Apps
112
Apps
Customer Apps
114
116
118
120
122
128
126
124
Reports / Dashboards
Login / SSO Service (OpenID Connect)
Federation Service (SAML) 130
API Token Service (OAuth) 132
Directory Service (SCIM) 134
Provisioning Service (SCIM / ATOM) 136
Event Service (REST) 138
RBAC Service (SCIM) 140
One Admin Console
Identity Platform
My Apps
API
Oracle Identity Cloud Service
142

Fig. 1

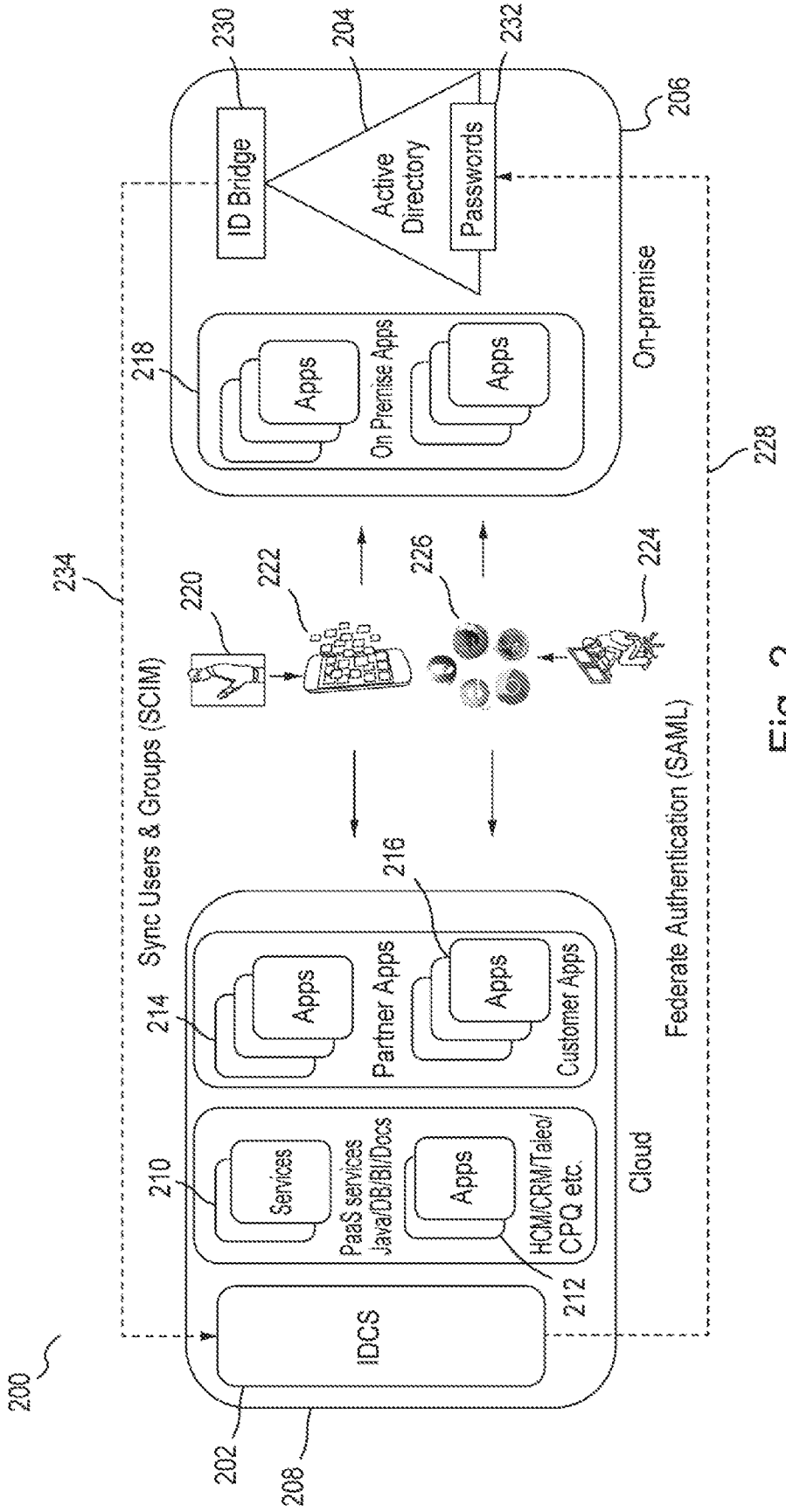
200
202
208
IDCS
212
Services
PaaS services
Java/DB/BI/Docs
210
Apps
HCM/CRM/Taleo/
CPQ etc.
Cloud
214
Apps
Partner Apps
216
Apps
Customer Apps
Federate Authentication (SAML)
234
Sync Users & Groups (SCIM)
220
222
226
224
228
218
Apps
On Premise Apps
Apps
On-premise
206
ID Bridge
230
Active
Directory
204
Passwords
232

Fig. 2

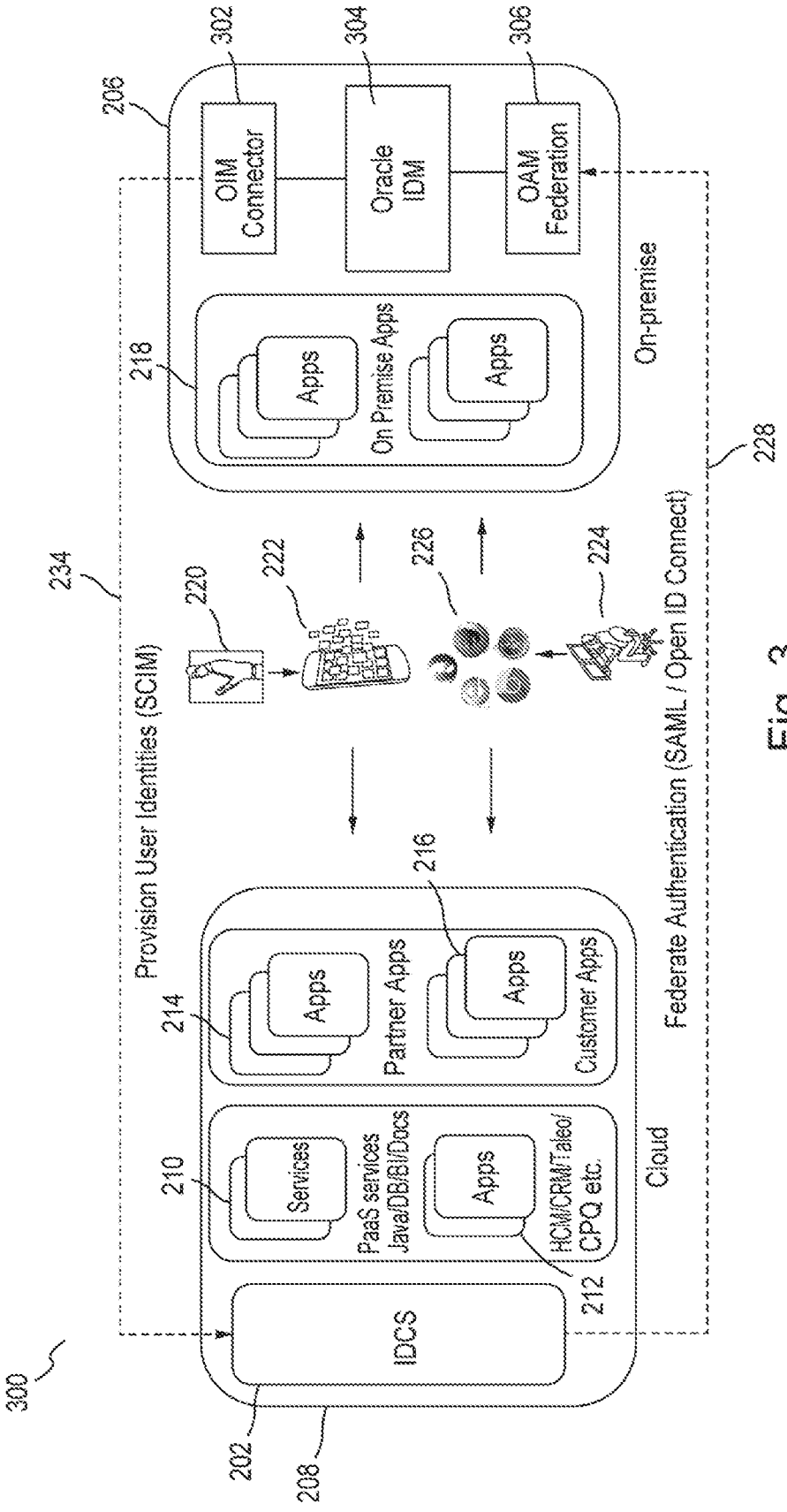
300
202
208
IDCS
212
Services
PaaS services
Java/DB/BI/Docs
210
Apps
HCM/CRM/Taleo/
CPQ etc.
Cloud
214
Apps
Partner Apps
216
Apps
Customer Apps
Provision User Identities (SCIM)
234
220
222
226
224
Federate Authentication (SAML / Open ID Connect)
228
206
218
Apps
On Premise Apps
Apps
On-premise
302
OIM
Connector
304
Oracle
IDM
306
OAM
Federation

Fig. 3

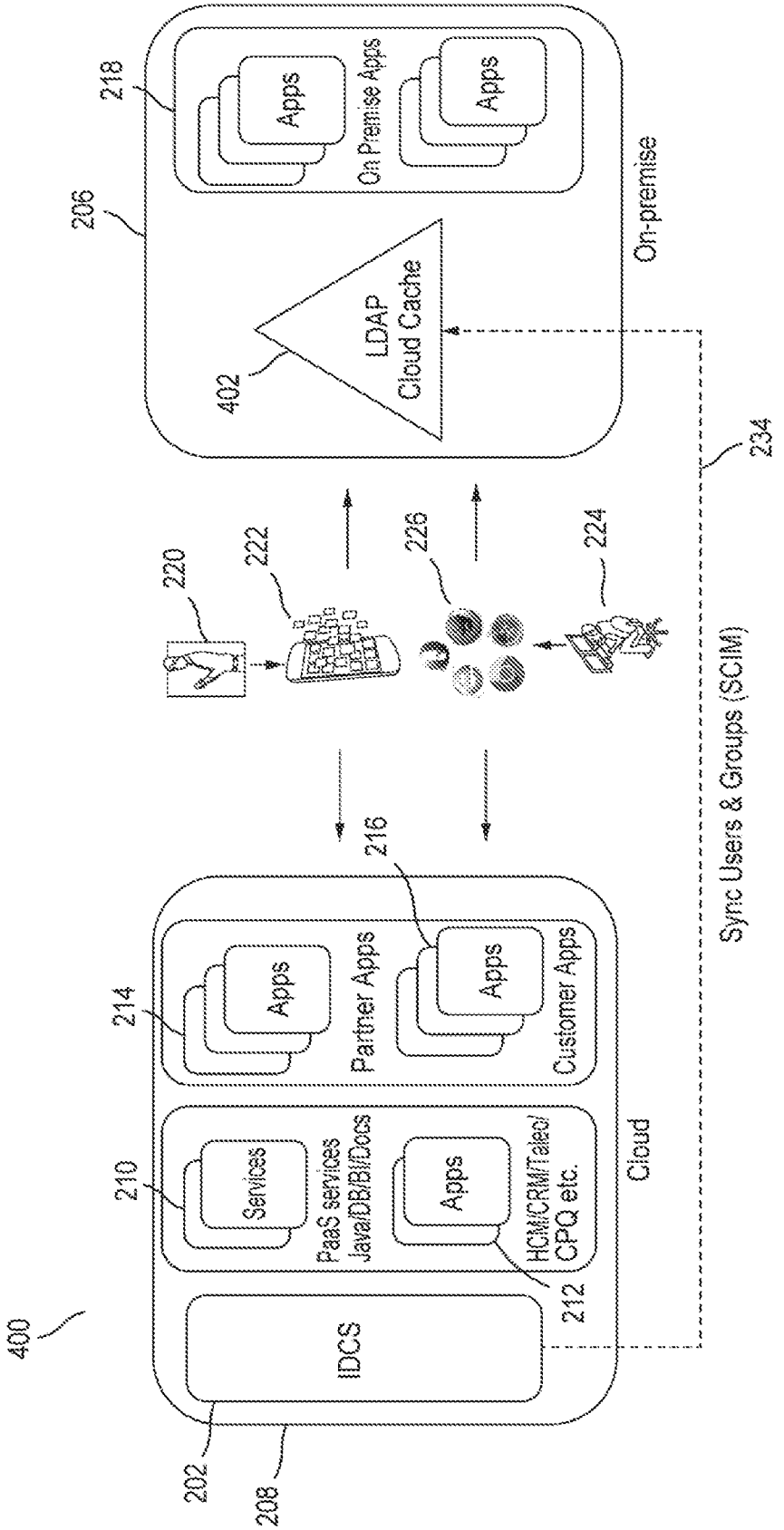
400
202
208
IDCS
212
Cloud
210
Services
PaaS services
Java/DB/BI/Docs
Apps
HCM/CRM/Taleo/
CPQ etc.
214
Apps
Partner Apps
216
Apps
Customer Apps
220
222
226
224
234
Sync Users & Groups (SCIM)
206
On-premise
218
Apps
On Premise Apps
Apps
402
LDAP
Cloud Cache

Fig. 4

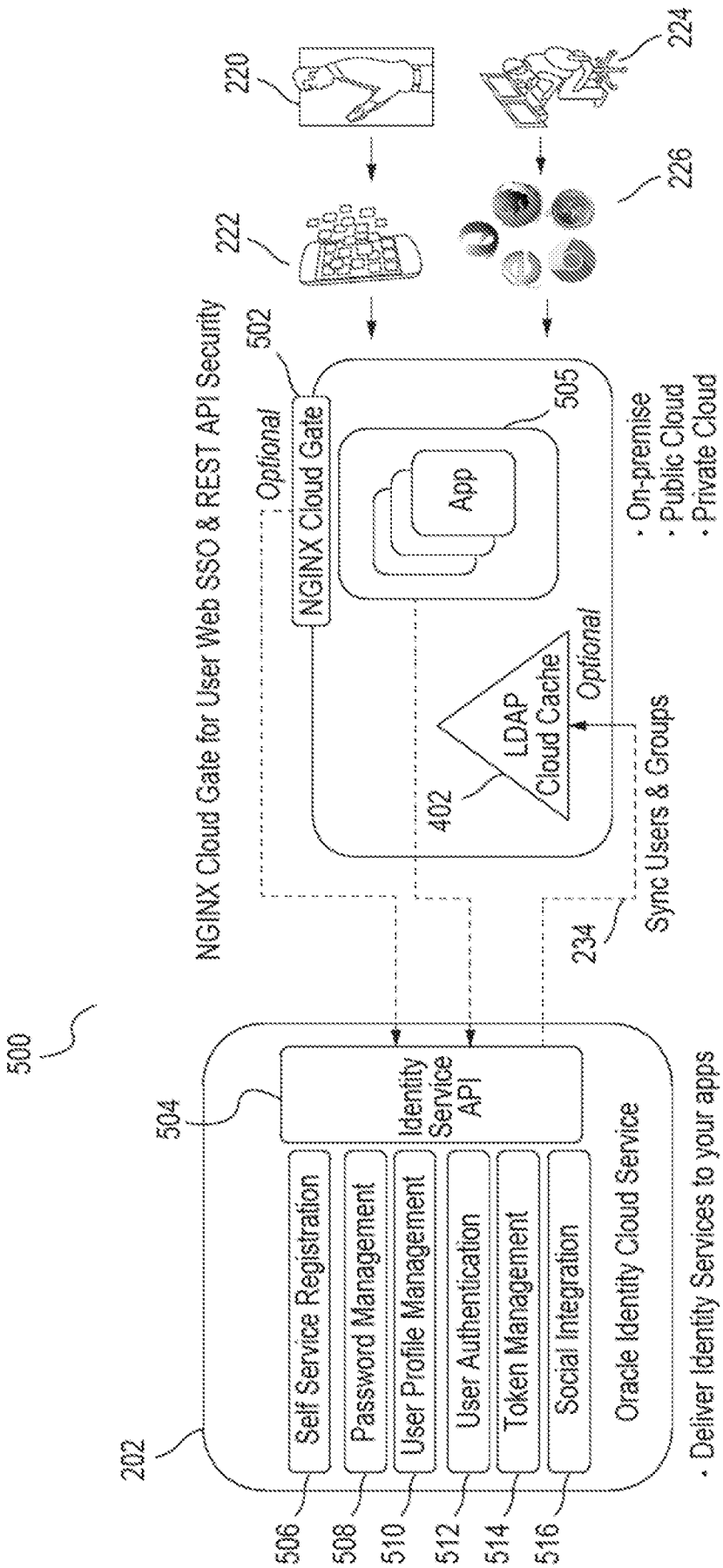
500
202
Oracle Identity Cloud Service
504
Identity Service API
Self Service Registration
Password Management
User Profile Management
User Authentication
Token Management
Social Integration
506
508
510
512
514
516
• Deliver Identity Services to your apps
• OpenID Connect, OAuth2, SAML2, SCIM Standards
• Self Service and Social Identity
234
Sync Users & Groups
NGINX Cloud Gate for User Web SSO & REST API Security
Optional
502
NGINX Cloud Gate
505
App
402
LDAP Cloud Cache
Optional
• On-premise
• Public Cloud
• Private Cloud
222
220
224
226

Fig. 5

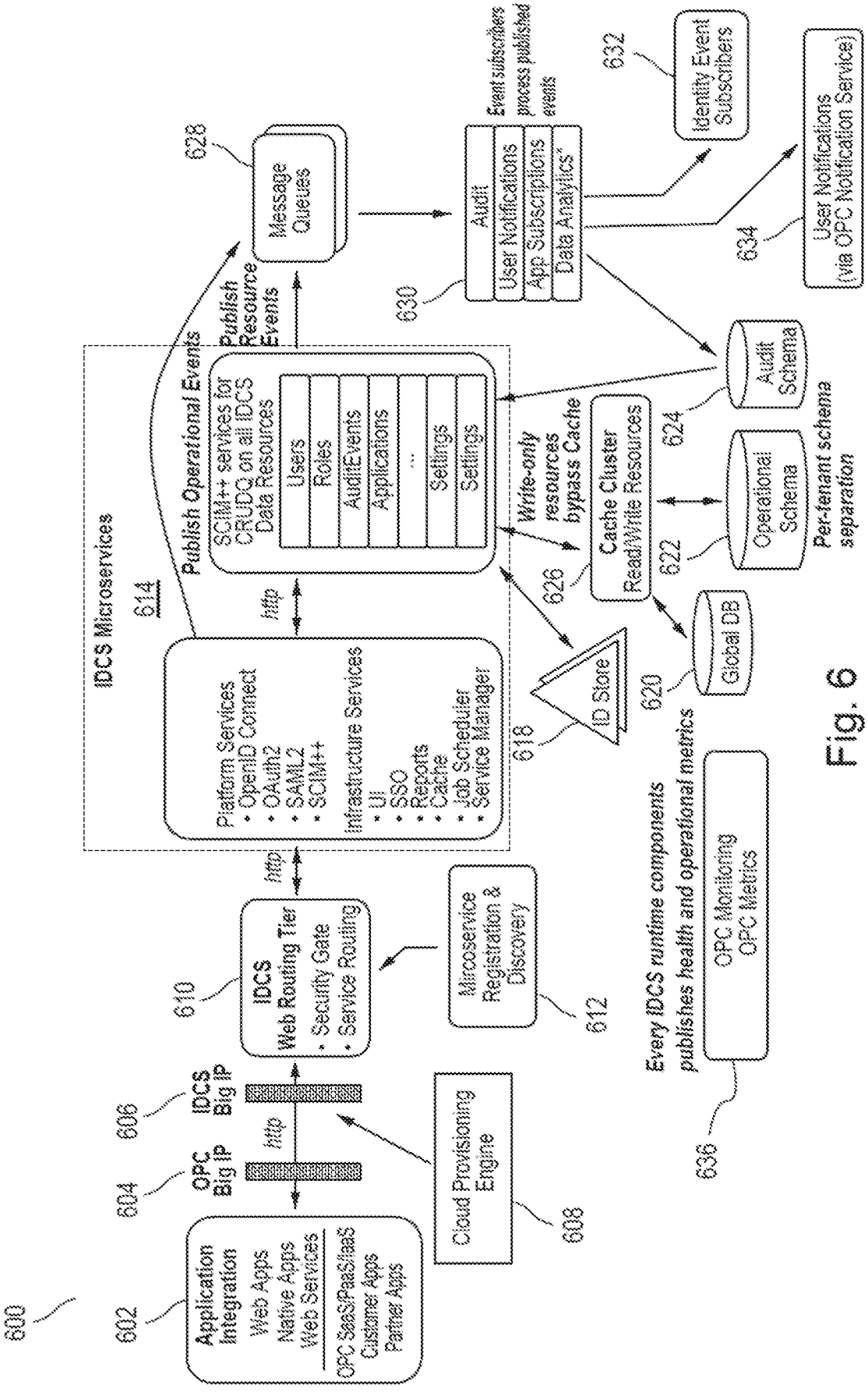
600
602
Application Integration
Web Apps
Native Apps
Web Services
OPC SaaS/PaaS/IaaS
Customer Apps
Partner Apps
604
OPC Big IP
606
IDCS Big IP
http
608
Cloud Provisioning Engine
610
IDCS Web Routing Tier
• Security Gate
• Service Routing
http
612
Mircoservice Registration & Discovery
IDCS Microservices
614
Platform Services
• OpenID Connect
• OAuth2
• SAML2
• SCIM++
Infrastructure Services
• UI
• SSO
• Reports
• Cache
• Job Scheduler
• Service Manager
http
Publish Operational Events
Publish Resource Events
SCIM++ services for CRUDQ on all IDCS Data Resources
Users
Roles
AuditEvents
Applications
...
Settings
Settings
628
Message Queues
630
Audit
User Notifications
App Subscriptions
Data Analytics*
Event subscribers process published events
632
Identity Event Subscribers
634
User Notifications (via OPC Notification Service)
Write-only resources bypass Cache
626
Cache Cluster
Read/Write Resources
622
Operational Schema
624
Audit Schema
Per-tenant schema separation
618
ID Store
620
Global DB
Every IDCS runtime components publishes health and operational metrics
636
OPC Monitoring
OPC Metrics

Fig. 6

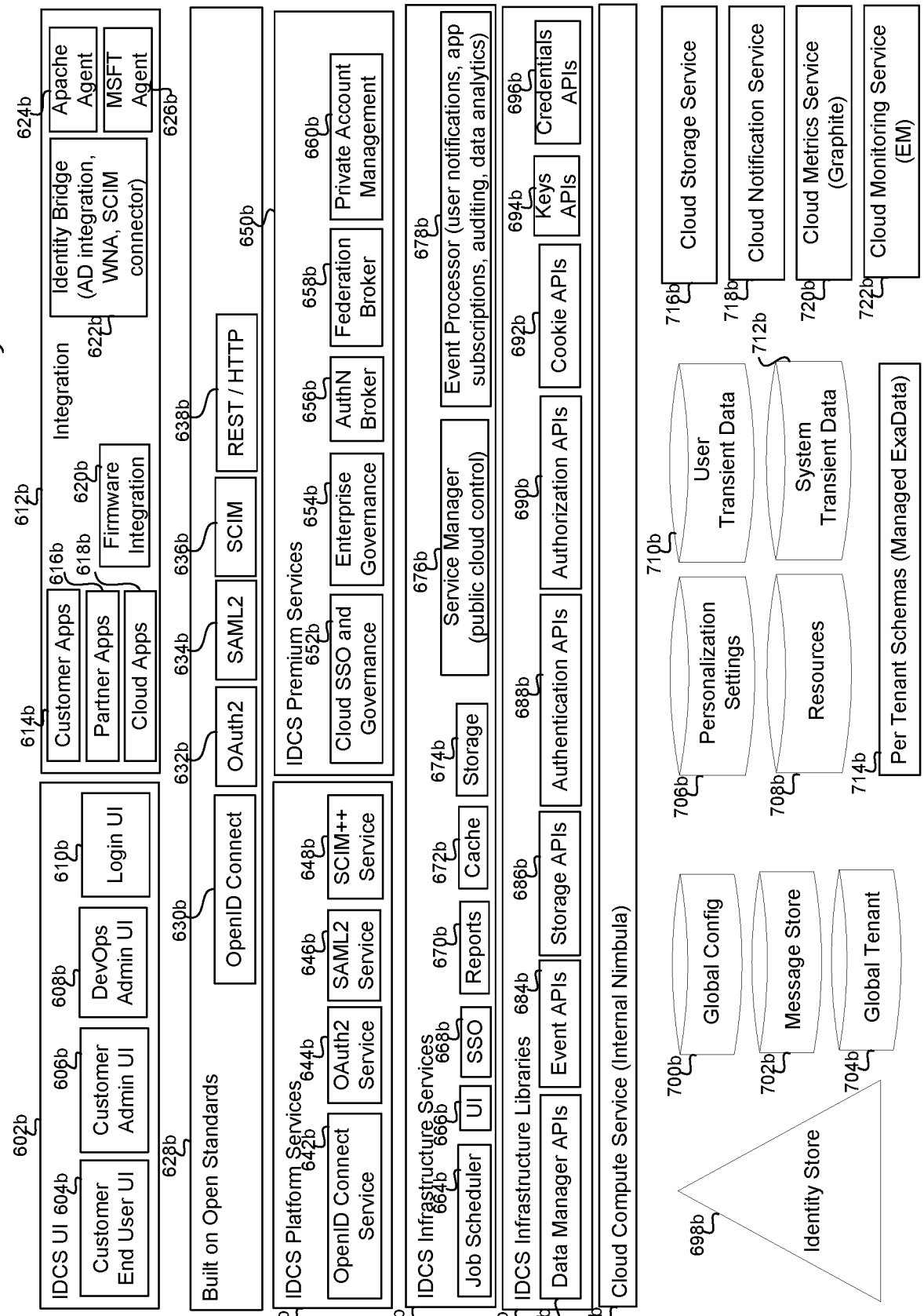
600b
602b
IDCS UI 604b
Customer End User UI
606b
Customer Admin UI
608b
DevOps Admin UI
610b
Login UI
628b
612b
Integration
614b
Customer Apps
616b
Partner Apps
618b
Cloud Apps
620b
Firmware Integration
622b
Identity Bridge (AD integration, WNA, SCIM connector)
624b
Apache Agent
626b
MSFT Agent
630b
Built on Open Standards
OpenID Connect
632b
OAuth2
634b
SAML2
636b
SCIM
638b
REST / HTTP
650b
640b
IDCS Platform Services
642b
OpenID Connect Service
644b
OAuth2 Service
646b
SAML2 Service
648b
SCIM++ Service
652b
IDCS Premium Services
Cloud SSO and Governance
654b
Enterprise Governance
656b
AuthN Broker
658b
Federation Broker
660b
Private Account Management
662b
IDCS Infrastructure Services
664b
Job Scheduler
666b
UI
668b
SSO
670b
Reports
672b
Cache
674b
Storage
676b
Service Manager (public cloud control)
678b
Event Processor (user notifications, app subscriptions, auditing, data analytics)
680b
IDCS Infrastructure Libraries
682b
Data Manager APIs
684b
Event APIs
686b
Storage APIs
688b
Authentication APIs
690b
Authorization APIs
692b
Cookie APIs
694b
Keys APIs
696b
Credentials APIs
602b
Cloud Compute Service (Internal Nimbula)
698b
Identity Store
700b
Global Config
702b
Message Store
704b
Global Tenant
706b
Personalization Settings
708b
Resources
710b
User Transient Data
712b
System Transient Data
714b
Per Tenant Schemas (Managed ExaData)
716b
Cloud Storage Service
718b
Cloud Notification Service
720b
Cloud Metrics Service (Graphite)
722b
Cloud Monitoring Service (EM)

Fig. 6A

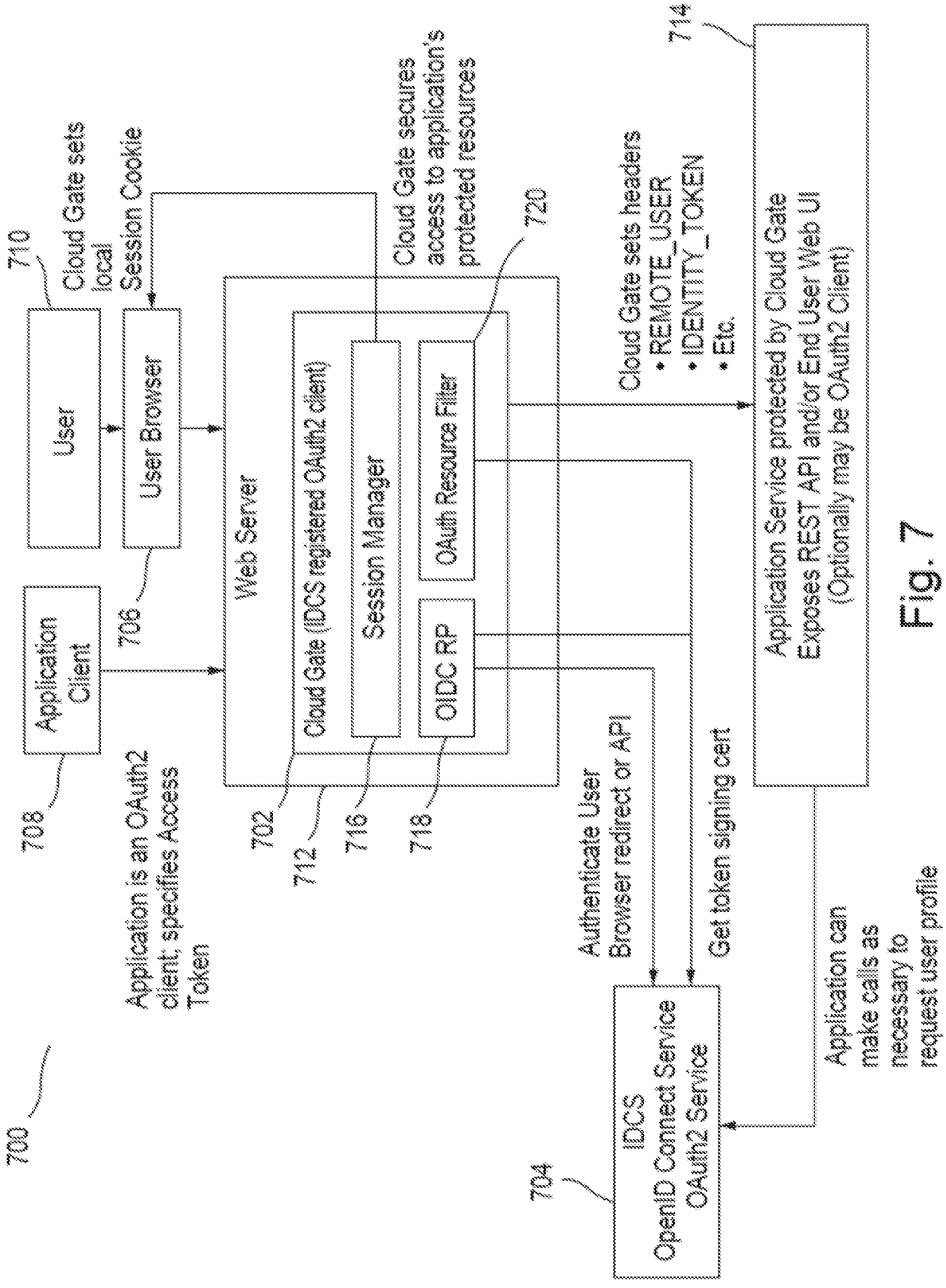
700
710
User
Cloud Gate sets local Session Cookie
User Browser
706
Application Client
708
Application is an OAuth2 client, specifies Access Token
Web Server
Cloud Gate secures access to application's protected resources
Cloud Gate (IDCS registered OAuth2 client)
Session Manager
OIDC RP
OAuth Resource Filter
720
702
712
716
718
Cloud Gate sets headers
• REMOTE_USER
• IDENTITY_TOKEN
• Etc.
714
Application Service protected by Cloud Gate
Exposes REST API and/or End User Web UI
(Optionally may be OAuth2 Client)
Authenticate User
Browser redirect or API
Get token signing cert
Application can make calls as necessary to request user profile
704
IDCS
OpenID Connect Service
OAuth2 Service

Fig. 7

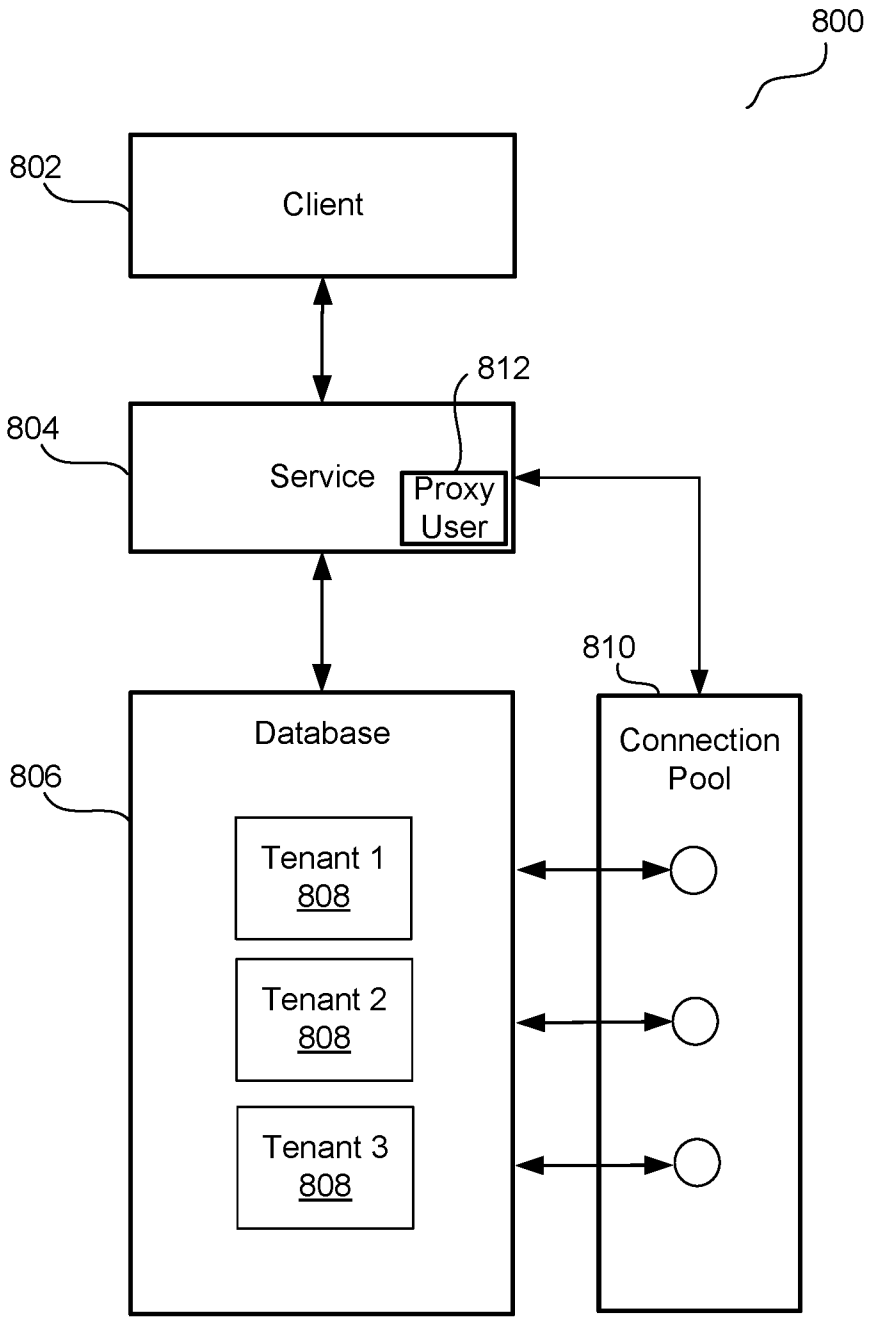
800
802
Client
812
804
Service
Proxy User
810
Database
Connection Pool
806
Tenant 1
808
Tenant 2
808
Tenant 3
808

Fig. 8

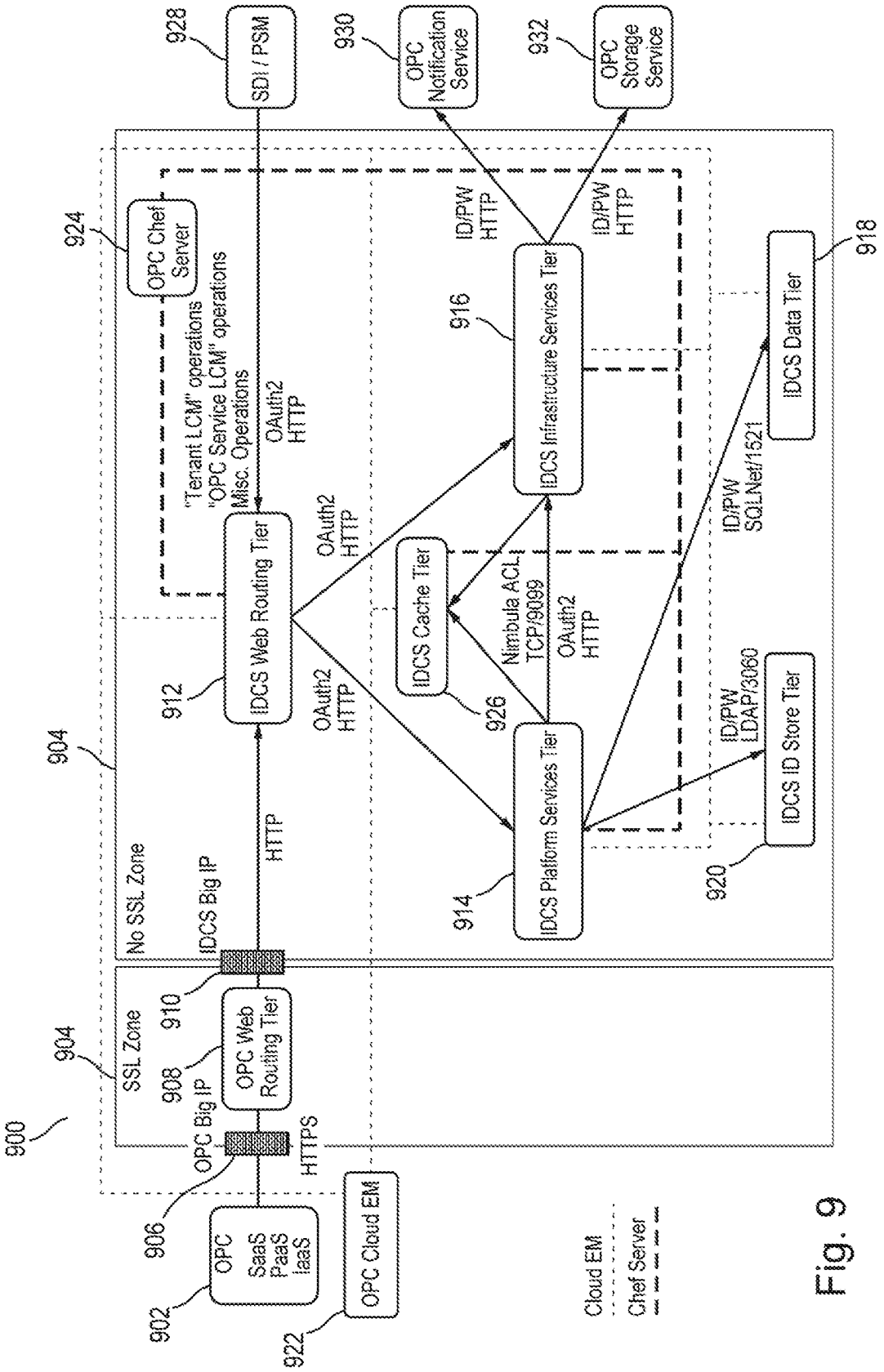
900
902
OPC
SaaS
PaaS
IaaS
904
SSL Zone
906
OPC Big IP
908
OPC Web Routing Tier
910
HTTPS
HTTP
IDCS Big IP
No SSL Zone
912
IDCS Web Routing Tier
914
IDCS Platform Services Tier
916
IDCS Infrastructure Services Tier
918
IDCS Data Tier
920
IDCS ID Store Tier
922
OPC Cloud EM
924
OPC Chef Server
926
IDCS Cache Tier
928
SDI / PSM
930
OPC Notification Service
932
OPC Storage Service
"Tenant LCM" operations
"OPC Service LCM" operations
Misc. Operations
OAuth2
HTTP
ID/PW
HTTP
ID/PW
SQLNet/1521
ID/PW
LDAP/3060
Nimbula ACL
TCP/9099
Cloud EM
Chef Server

Fig. 9

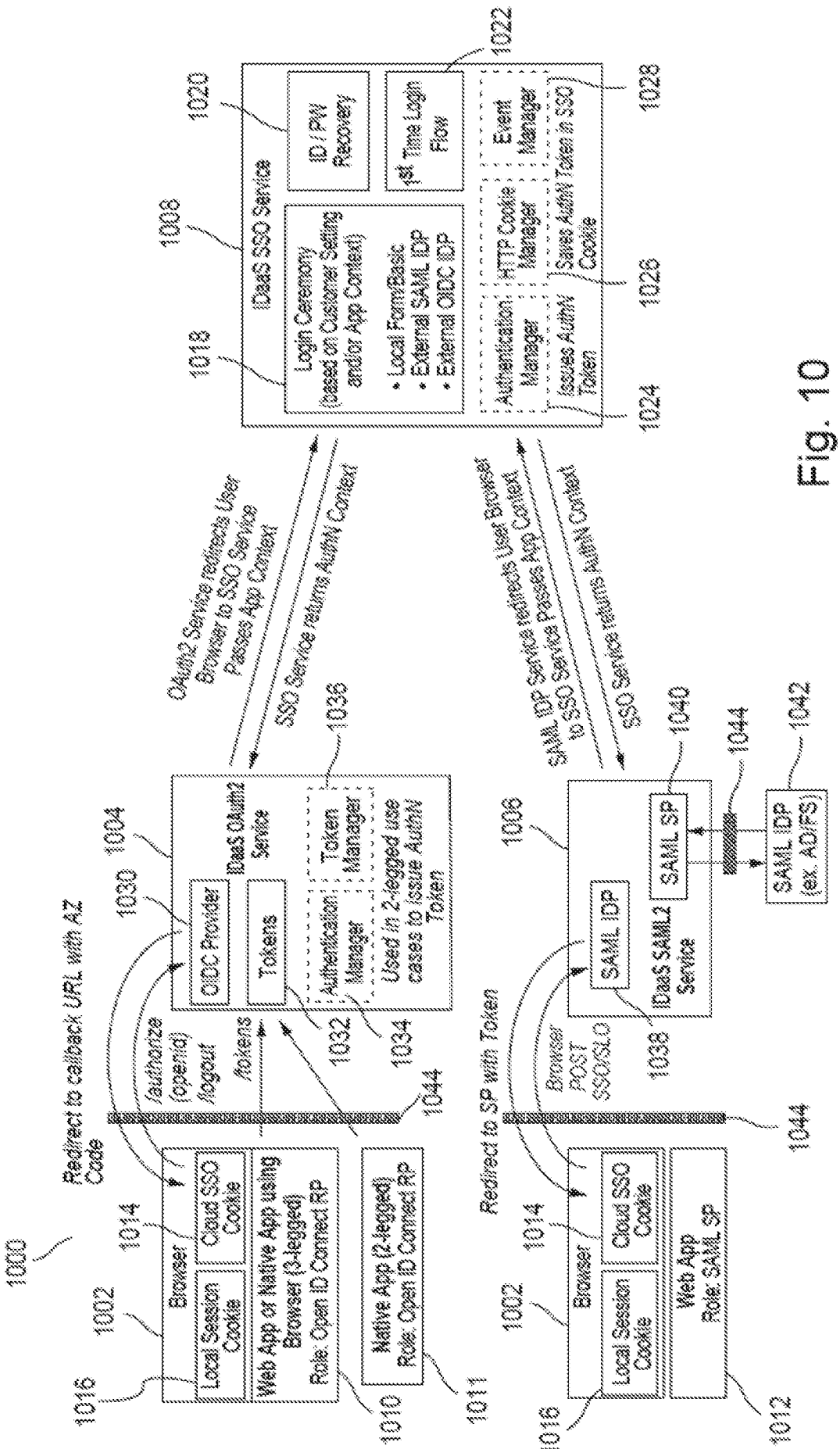
1000
Redirect to callback URL with AZ Code
1002
1014
1016
Browser
Local Session Cookie
Cloud SSO Cookie
Web App or Native App using Browser (3-legged)
Role: Open ID Connect RP
1010
Native App (2-legged)
Role: Open ID Connect RP
1011
/authorize (openid)
/logout
/tokens
1030
1004
DaaS OAuth2 Service
OIDC Provider
Tokens
Authentication Manager
Token Manager
Used in 2-legged use cases to issue AuthN Token
1032
1034
1036
1044
OAuth2 Service redirects User Browser to SSO Service Passes App Context
SSO Service returns AuthN Context
1006
DaaS SSO Service
Login Ceremony (based on Customer Setting and/or App Context)
• Local Form/Basic
• External SAML IDP
• External OIDC IDP
ID / PW Recovery
1st Time Login Flow
Authentication Manager
Issues AuthN Token
HTTP Cookie Manager
Saves AuthN Token in SSO Cookie
Event Manager
1018
1020
1022
1024
1026
1028
SAML IDP Service redirects User Browser to SSO Service Passes App Context
SSO Service returns AuthN Context
1040
1042
1044
SAML SP
SAML IDP (ex. AD/FS)
Redirect to SP with Token
Browser POST SSO/SLO
SAML IDP
DaaS SAML2 Service
1038
Web App
Role: SAML SP
1012

Fig. 10

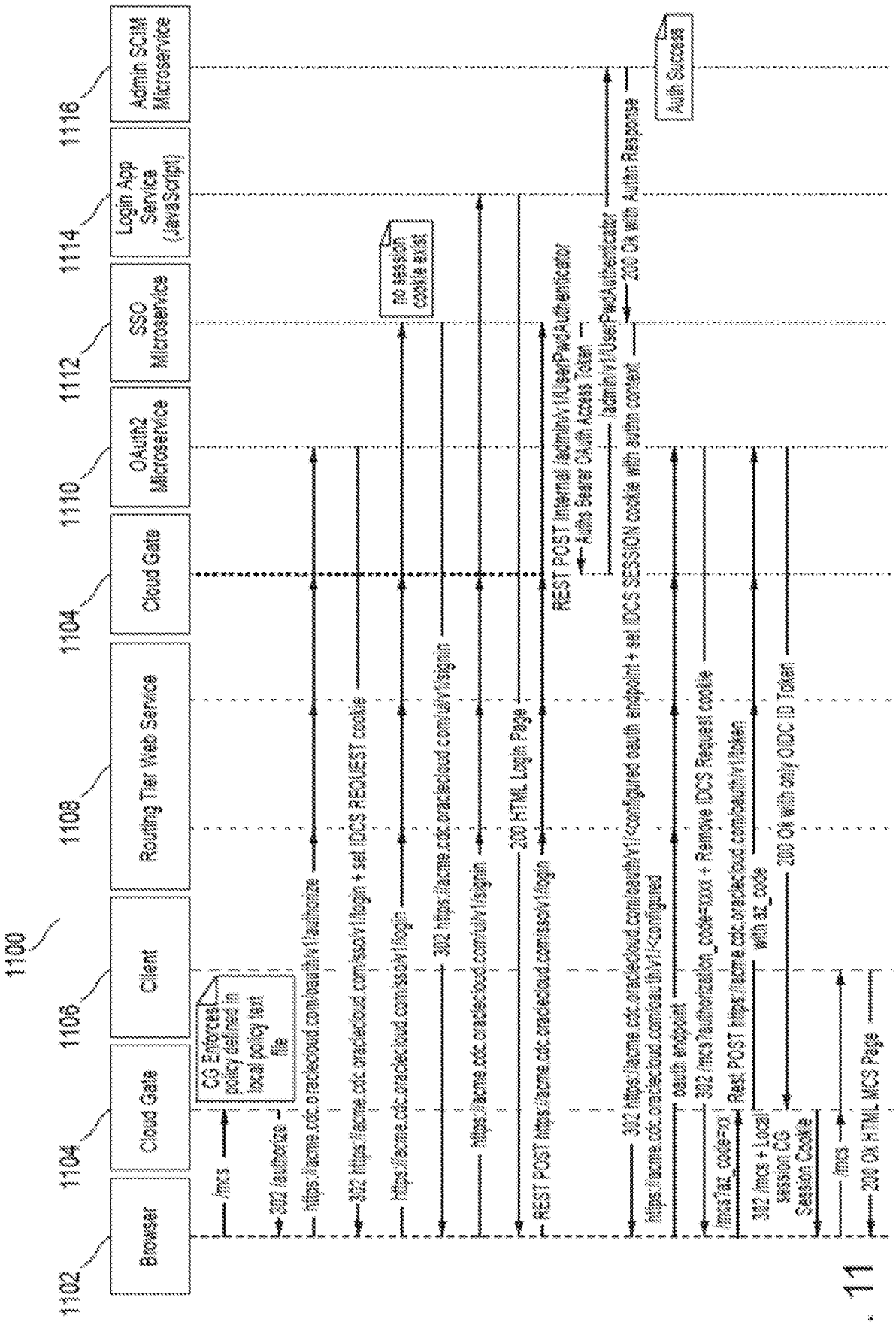
1100
1102
Browser
1104
Cloud Gate
1106
Client
1108
Routing Tier Web Service
1104
Cloud Gate
1110
OAuth2 Microservice
1112
SSO Microservice
1114
Login App Service (JavaScript)
1116
Admin SCIM Microservice
/mcs
CG Enforces policy defined in local policy text file
302 /authorize
https://acme.cdc.oraclecloud.com/oauth/v1/authorize
302 https://acme.cdc.oraclecloud.com/sso/v1/login + set IDCS REQUEST cookie
https://acme.cdc.oraclecloud.com/sso/v1/login
no session cookie exist
302 https://acme.cdc.oraclecloud.com/ui/v1/signin
https://acme.cdc.oraclecloud.com/ui/v1/signin
200 HTML Login Page
REST POST https://acme.cdc.oraclecloud.com/sso/v1/login
REST POST Internal /admin/v1/UserPwdAuthenticator
Authn Bearer OAuth Access Token
/admin/v1/UserPwdAuthenticator
200 Ok with Authn Response
Auth Success
302 https://acme.cdc.oraclecloud.com/oauth/v1/<configured oauth endpoint> + set IDCS SESSION cookie with authn context
https://acme.cdc.oraclecloud.com/oauth/v1/<configured oauth endpoint>
302 /mcs?authorization_code=xxx + Remove IDCS Request cookie
/mcs?az_code=xx
Rest POST https://acme.cdc.oraclecloud.com/oauth/v1/token with az_code
302 /mcs + Local session CG Session Cookie
200 Ok with only OIDC ID Token
/mcs
200 Ok HTML MCS Page

Fig. 11

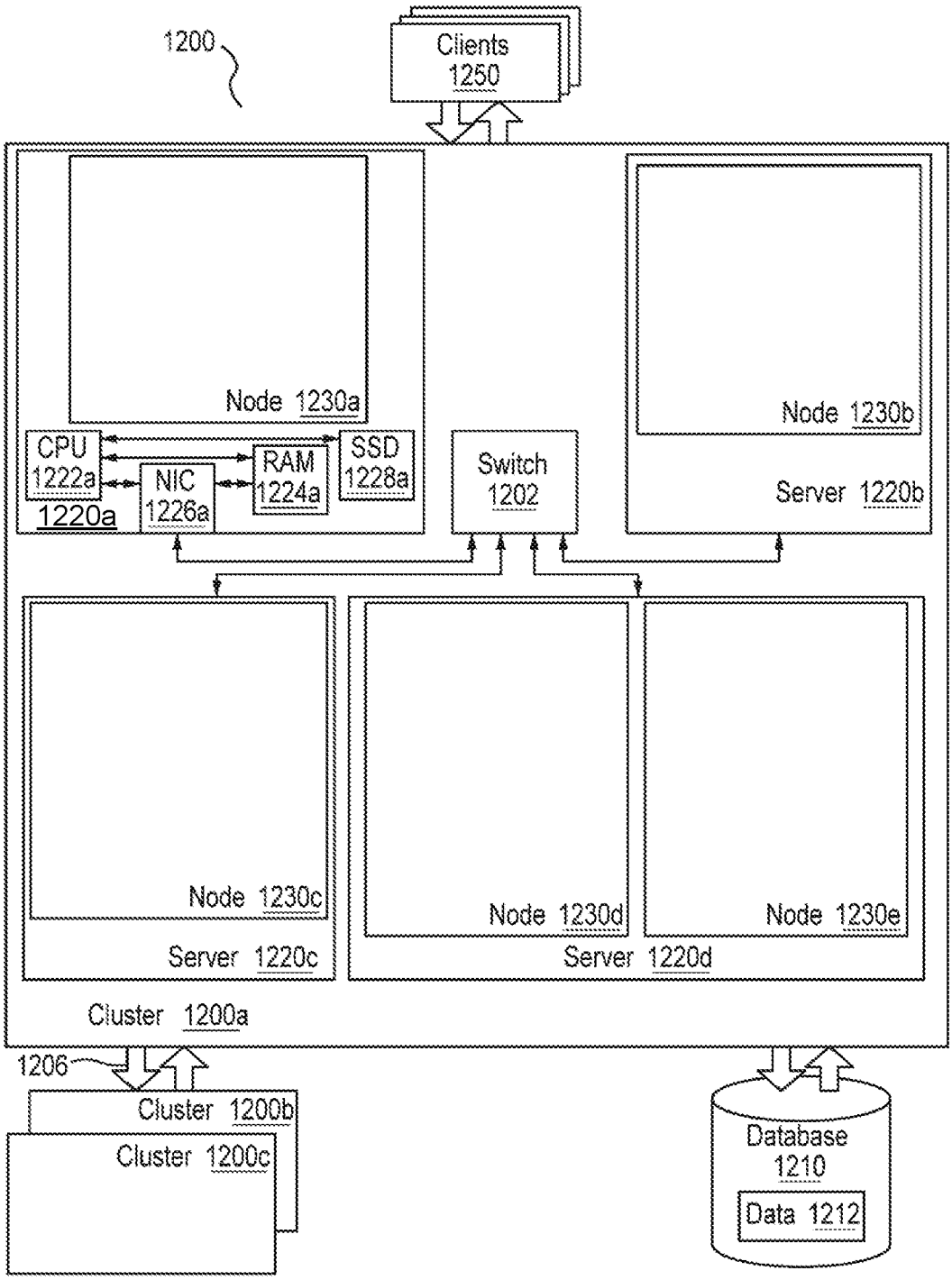
1200
Clients
1250
Node 1230a
CPU
1222a
NIC
1226a
RAM
1224a
SSD
1228a
1220a
Switch
1202
Node 1230b
Server 1220b
Node 1230c
Server 1220c
Node 1230d
Node 1230e
Server 1220d
Cluster 1200a
1206
Cluster 1200b
Cluster 1200c
Database
1210
Data 1212

Fig. 12

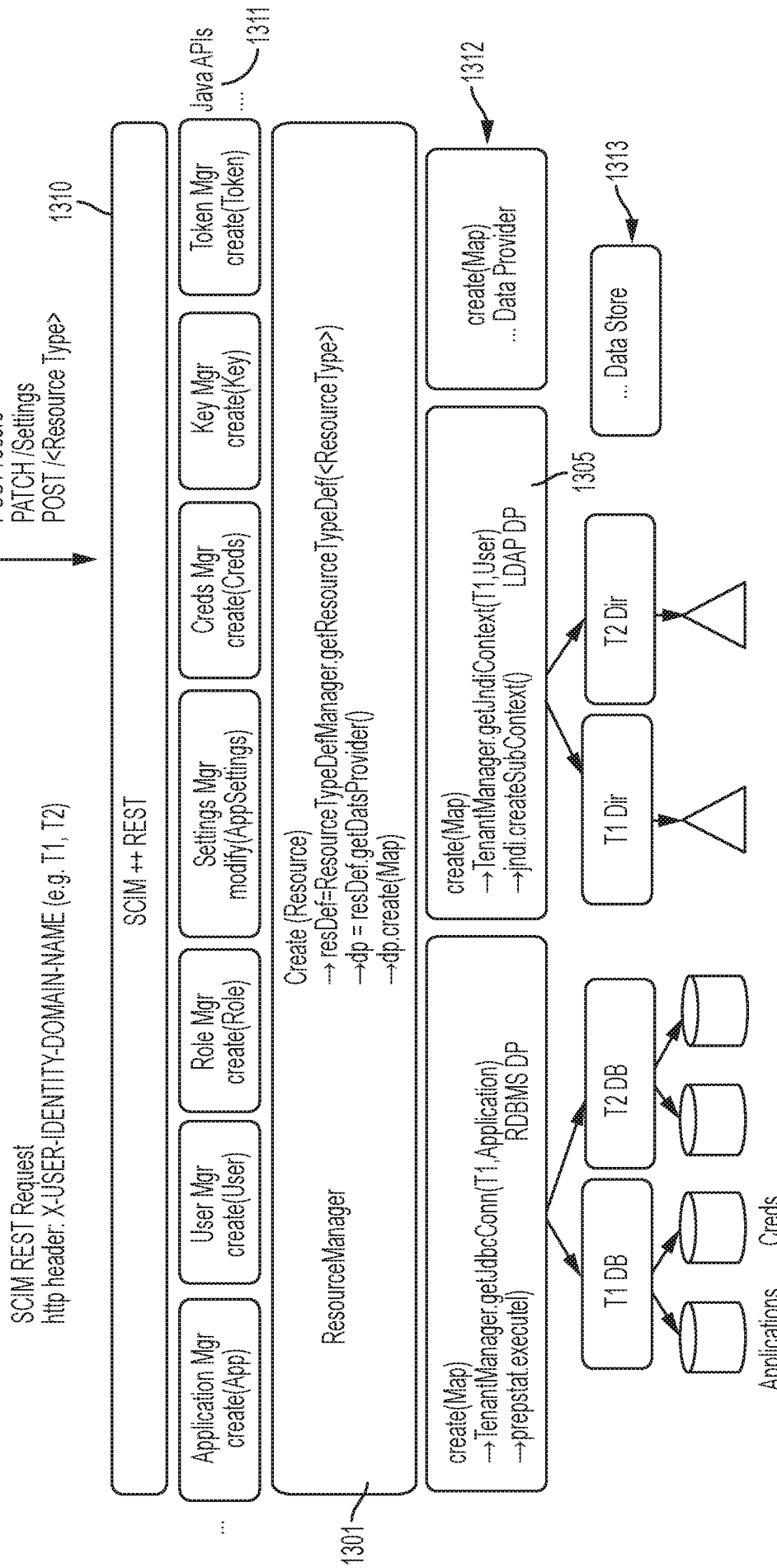
SCIM REST Request
http header: X-USER-IDENTITY-DOMAIN-NAME (e.g. T1, T2)
POST /Applications
POST /Users
PATCH /Settings
POST /<Resource Type>
1310
SCIM ++ REST
Application Mgr
create(App)
User Mgr
create(User)
Role Mgr
create(Role)
Settings Mgr
modify(AppSettings)
Creds Mgr
create(Creds)
Key Mgr
create(Key)
Token Mgr
create(Token)
Java APIs
1311
ResourceManager
Create (Resource)
→ resDef=ResourceTypeDefManager.getResourceTypeDef(<ResourceType>)
→dp = resDef.getDatsProvider()
→dp.create(Map)
1301
create(Map)
→TenantManager.getJdbcConn(T1,Application)
→prepstat.execute()
RDBMS DP
create(Map)
→TenantManager.getJndiContext(T1,User)
→jndi.createSubContext()
LDAP DP
1305
create(Map)
... Data Provider
1312
T1 DB
T2 DB
T1 Dir
T2 Dir
... Data Store
1313
Applications
Creds

Fig. 13

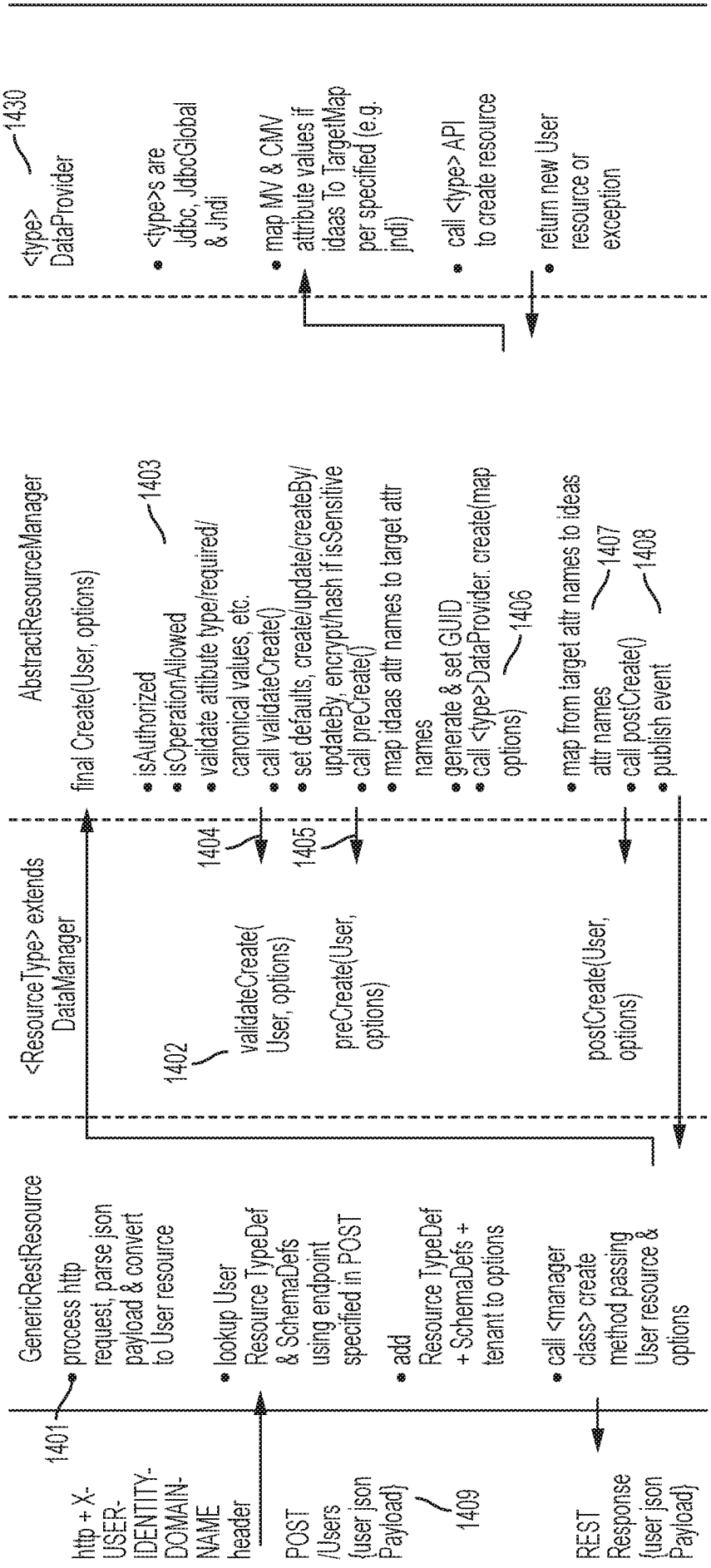
GenericRestResource
<ResourceType> extends DataManager
AbstractResourceManager
<type> DataProvider
1430
1401
http + X-USER-IDENTITY-DOMAIN-NAME header
process http request, parse json payload & convert to User resource
lookup User Resource TypeDef & SchemaDefs using endpoint specified in POST
1402
validateCreate(User, options)
final Create(User, options)
isAuthorized
isOperationAllowed
validate attibute type/required/ canonical values, etc.
1403
call validateCreate()
1404
set defaults, create/update/createBy/ updateBy, encrypt/hash if isSensitive
call preCreate()
1405
preCreate(User, options)
map idaas attr names to target attr names
generate & set GUID
call <type>DataProvider. create(map options)
1406
POST /Users {user json Payload}
1409
add Resource TypeDef + SchemaDefs + tenant to options
call <manager class> create method passing User resource & options
map from target attr names to ideas attr names
1407
call postCreate()
1408
publish event
postCreate(User, options)
REST Response {user json Payload}
<type>s are Jdbc, JdbcGlobal & Jndi
map MV & CMV attribute values if idaas To TargetMap per specified (e.g. jndi)
call <type> API to create resource
return new User resource or exception

FIG. 14

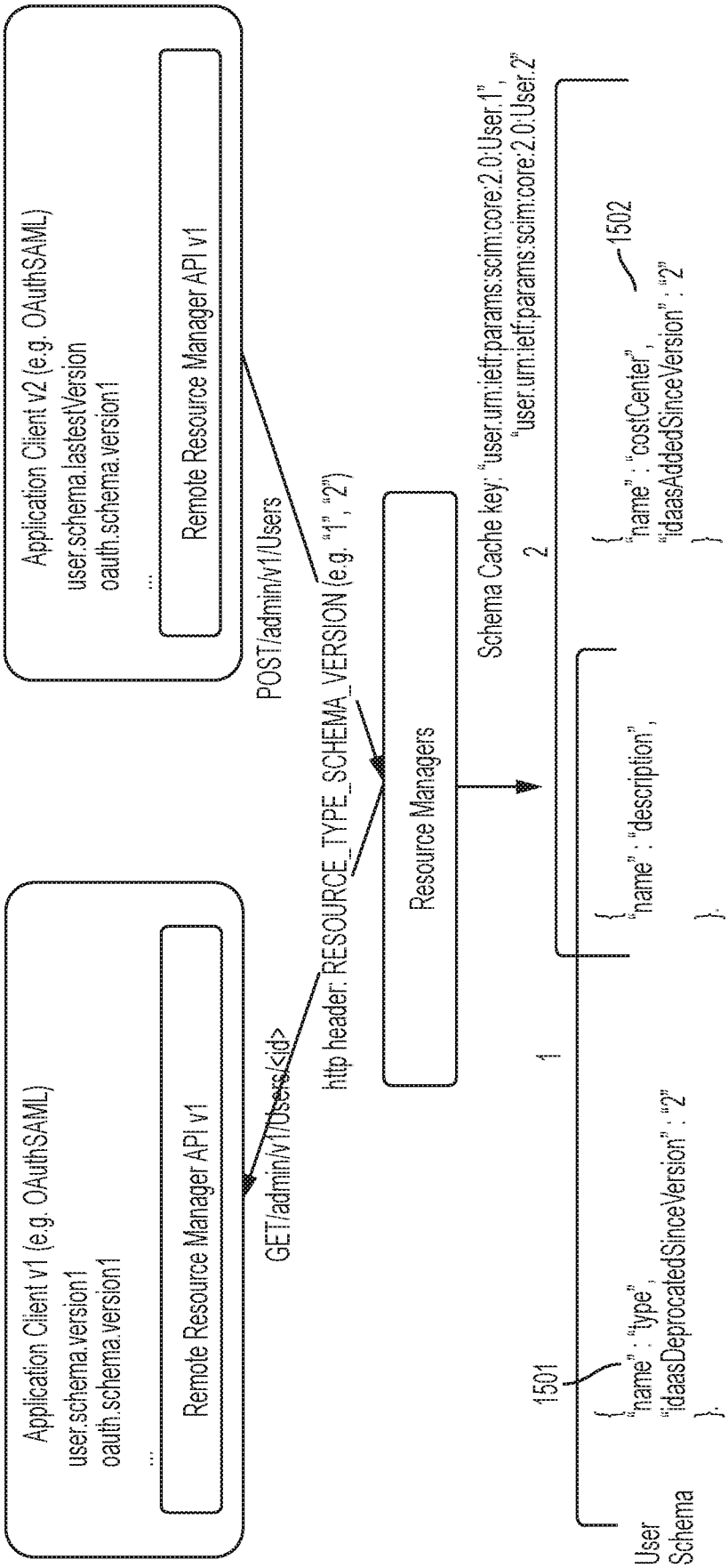
Application Client v2 (e.g. OAuthSAML)
user.schema.lastestVersion
oauth.schema.version1
...
Remote Resource Manager API v1
POST/admin/v1/Users
Application Client v1 (e.g. OAuthSAML)
user.schema.version1
oauth.schema.version1
...
Remote Resource Manager API v1
GET/admin/v1/Users/<id>
http header: RESOURCE_TYPE_SCHEMA_VERSION (e.g. "1", "2")
Resource Managers
Schema Cache key: "user.urn:ietf:params:scim:core:2.0:User.1",
"user.urn:ietf:params:scim:core:2.0:User.2"
1
2
1501
1502
User Schema
{
"name" : "type",
"idaasDeprocatedSinceVersion" : "2"
},
{
"name" : "description",
},
{
"name" : "costCenter",
"idaasAddedSinceVersion" : "2"
}

Fig. 15

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2017/047726

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F21/41 G06F21/60
ADD.

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

B. FIELDS SEARCHED

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

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

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2016/124742 A1 (RANGASAMY VENKATACHALAM [US] ET AL) 5 May 2016 (2016-05-05) paragraph [0041] - paragraph [0106]; figures 1-11	1-20
X	US 2014/280948 A1 (SCHMIDT RENE W [DK] ET AL) 18 September 2014 (2014-09-18) paragraph [0013] - paragraph [0054]	1-20
X	US 2014/090037 A1 (SINGH SERVESH PRATAP [IN]) 27 March 2014 (2014-03-27) paragraph [0021] - paragraph [0051]	1-20
X	US 2013/031136 A1 (SHAH ANKUR B [IN]) 31 January 2013 (2013-01-31) paragraph [0101] - paragraph [0107]	1-20

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

☒ See patent family annex.

* Special categories of cited documents :

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

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

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

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

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

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

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

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

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
19 October 2017	07/11/2017

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

1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2017/047726

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2016124742 A1	05-05-2016	AU 2015338902 A1	22-12-2016
		CA 2951939 A1	06-05-2016
		CN 106464736 A	22-02-2017
		EP 3155524 A1	19-04-2017
		JP 2017525232 A	31-08-2017
		SG 11201610099P A	30-05-2017
		US 2016124742 A1	05-05-2016
		US 2016127254 A1	05-05-2016
		US 2016127454 A1	05-05-2016
		US 2017111220 A1	20-04-2017
		WO 2016070145 A1	06-05-2016
US 2014280948 A1	18-09-2014	NONE	
US 2014090037 A1	27-03-2014	US 2014090037 A1	27-03-2014
		WO 2014046857 A1	27-03-2014
US 2013031136 A1	31-01-2013	NONE	