US012386635B2

(12) United States Patent
Mysore Jagadeesh et al.

(10) Patent No.: US 12,386,635 B2
(45) Date of Patent: Aug. 12, 2025

(54) **DATA MANAGEMENT TECHNIQUES FOR CLOUD REGIONS**

(71) Applicant: **Oracle International Corporation**, Redwood Shores, CA (US)

(72) Inventors: **Kavyashree Mysore Jagadeesh**, Seattle, WA (US); **Erik Joseph Miller**, Seattle, WA (US)

(73) Assignee: **Oracle International Corporation**, Redwood Shores, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 170 days.

(21) Appl. No.: **18/163,251**

(22) Filed: **Feb. 1, 2023**

(65) **Prior Publication Data**

US 2023/0251876 A1 Aug. 10, 2023

Related U.S. Application Data

(60) Provisional application No. 63/315,024, filed on Feb. 28, 2022, provisional application No. 63/312,814, filed on Feb. 22, 2022, provisional application No. 63/308,003, filed on Feb. 8, 2022.

(51) **Int. Cl.**
G06F 9/445 (2018.01)
G06F 9/4401 (2018.01)

(52) **U.S. Cl.**
CPC ***G06F 9/4451*** (2013.01); ***G06F 9/4416*** (2013.01)

(58) **Field of Classification Search**
CPC G06F 9/4451; G06F 9/4416; G06F 8/65
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

11,853,802 B1 * 12/2023 Wei G06F 9/45558
2013/0139139 A1 * 5/2013 Mallur G06F 8/65 717/170
2013/0232480 A1 9/2013 Winterfeldt et al.
2021/0067607 A1 * 3/2021 Gardner H04L 41/082

FOREIGN PATENT DOCUMENTS

WO 2021150366 A1 7/2021

OTHER PUBLICATIONS

International Application No. PCT/US2023/012312, International Search Report and the Written Opinion mailed on May 8, 2023, 11 pages.

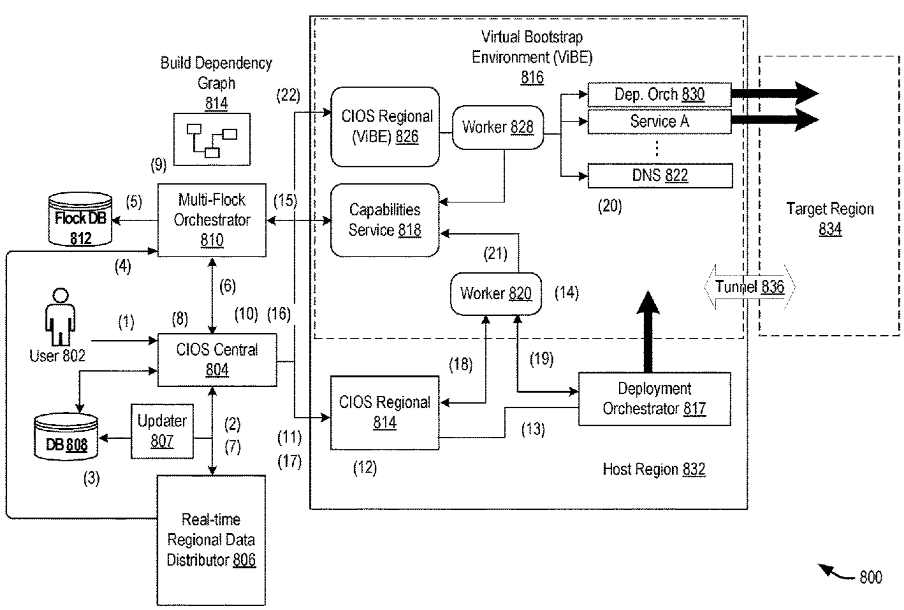
(Continued)

Primary Examiner — Jaweed A Abbaszadeh
Assistant Examiner — Gayathri Sampath
(74) *Attorney, Agent, or Firm* — Kilpatrick Townsend & Stockton LLP

(57) **ABSTRACT**

Techniques are described for performing an automated region build with real time region data. Region data including region identifiers and execution target identifiers for the region may be maintained. When a modification of the region data is detected (or new region data is detected), configuration files corresponding to bootstrapping resources (e.g., at the execution targets) within the region may be obtained. Operations are executed to cause the configuration files to be updated. This may include recompiling or otherwise injecting region data into the configuration files. A region build may be executed to bootstrap resources within the region using the updated configuration files.

20 Claims, 14 Drawing Sheets



(56) **References Cited**

OTHER PUBLICATIONS

International Application No. PCT/US2023/012312, "International Preliminary Report on Patentability", mailed Aug. 22, 2024, 9 pages.
International Application No. PCT/US2023/012312, "International Search Report and the Written Opinion", mailed May 8, 2023, 11 pages.

* cited by examiner

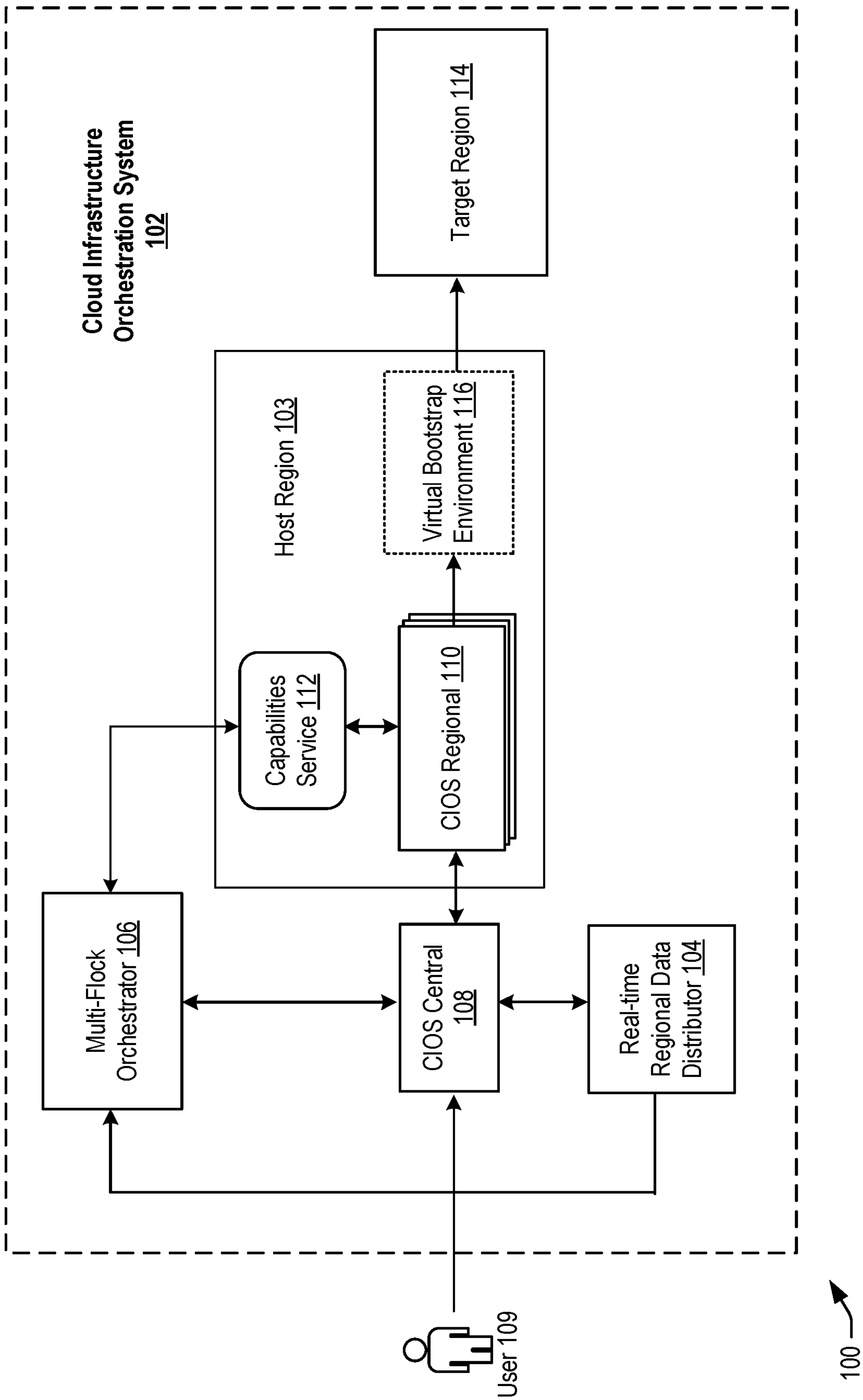

Cloud Infrastructure Orchestration System 102
Target Region 114
Host Region 103
Virtual Bootstrap Environment 116
Capabilities Service 112
CIOS Regional 110
Multi-Flock Orchestrator 106
CIOS Central 108
Real-time Regional Data Distributor 104
User 109
100

FIG. 1

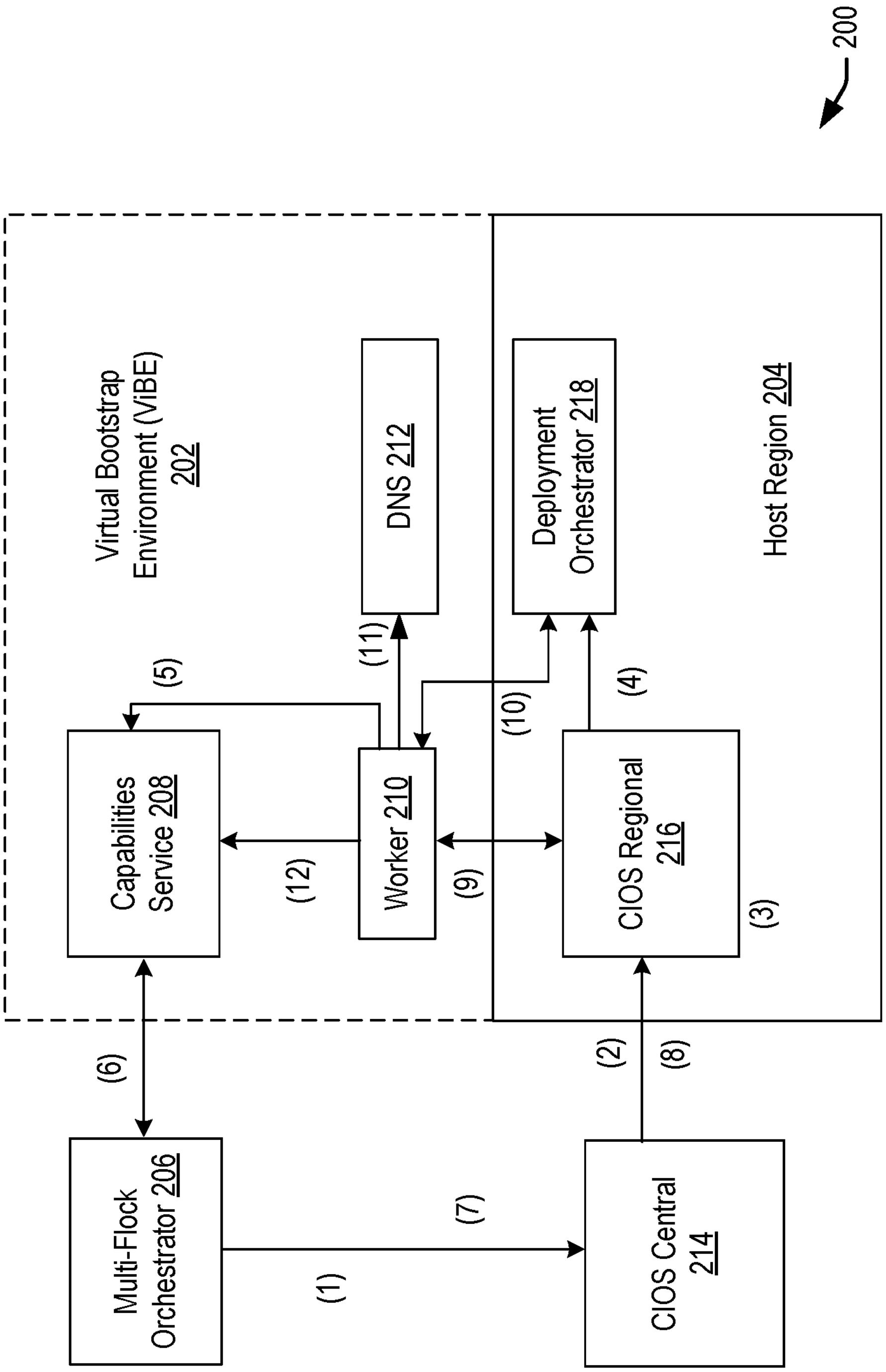

200
Virtual Bootstrap Environment (ViBE) 202
DNS 212
Deployment Orchestrator 218
Host Region 204
Capabilities Service 208
Worker 210
CIOS Regional 216
Multi-Flock Orchestrator 206
CIOS Central 214
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)

FIG. 2

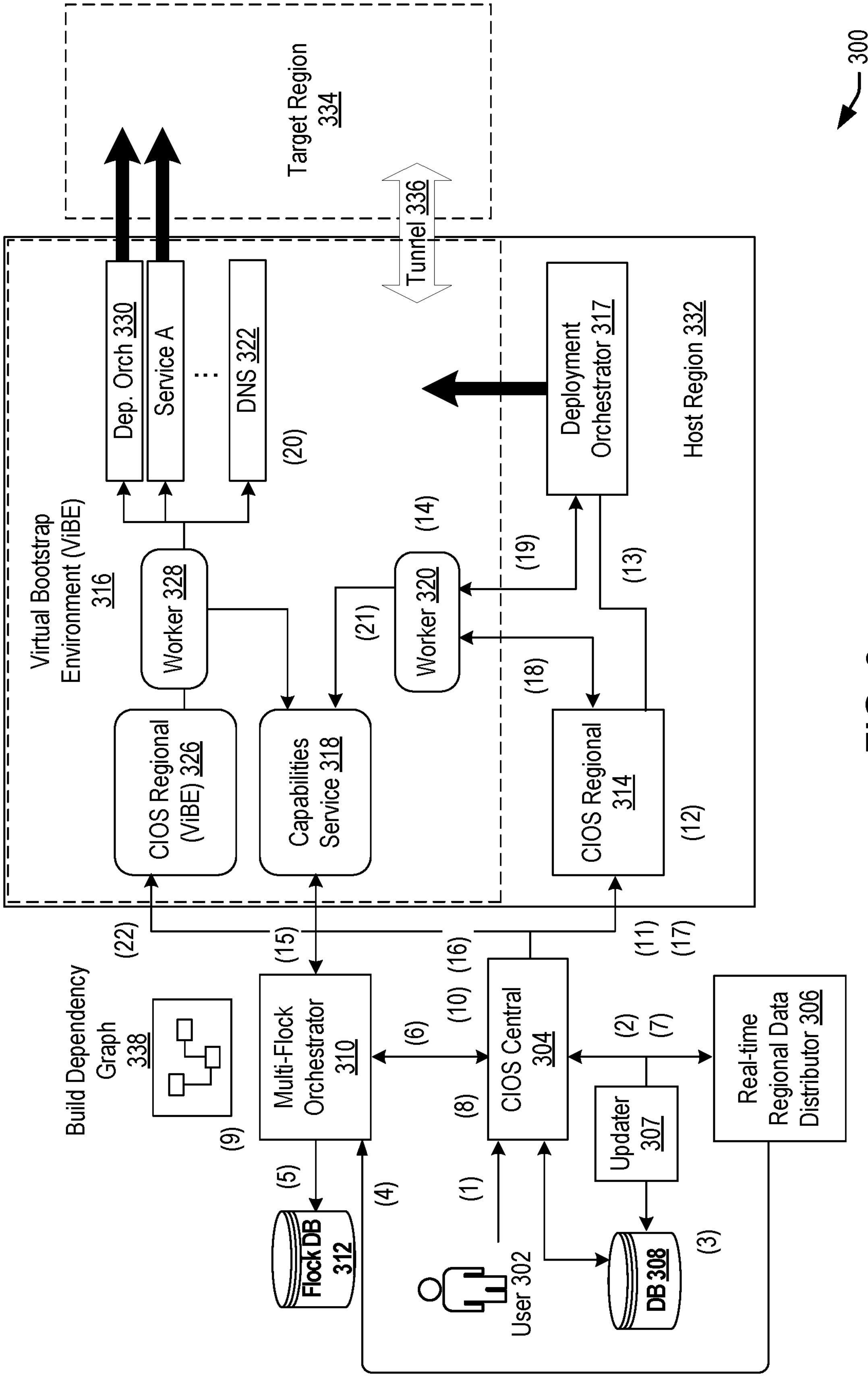

Target Region 334
Tunnel 336
Virtual Bootstrap Environment (ViBE) 316
Dep. Orch 330
Service A
...
DNS 322
(20)
Worker 328
(14)
Worker 320
(19)
(13)
(21)
(18)
Deployment Orchestrator 317
Host Region 332
CIOS Regional (ViBE) 326
Capabilities Service 318
CIOS Regional 314
(12)
(22)
(15)
(16)
(11)
(17)
Build Dependency Graph 338
Multi-Flock Orchestrator 310
(6)
(10)
(8)
CIOS Central 304
(2)
(7)
Real-time Regional Data Distributor 306
Updater 307
(9)
(5)
(4)
(1)
Flock DB 312
User 302
DB 308
(3)
300

FIG. 3

Region Data
400

```
locals {  <- 401
 realms = [                                   402
  { name: "realm1"                       410
    enum_name: "REALM1" },
  { name: "realm2"                       412
    enum_name: "REALM2" },  ]

 regions = [                                  404
  { realm: "realm1"                      414
    internal_name: "r1"
    public_name: "us-seattle-1"
    ...},
  { realm: "realm2"                      416
    internal_name: "r2"
    public_name: "us-phoenix-1"
    ...},  ]

 all_regions = [                              406
  { realm: "realm1"                      418
    internal_name: "r1"
    public_name: "us-seattle-1"
    ...
    state: "Production",
    ViBE_region: ""},
  { realm: "realm2"                      420
    internal_name: "r2"
    public_name: "us-phoenix-1"
    ...
    state: "Production",
    ViBE_region: ""}, ]

 ads = [                                      408
  { name: "sea-ad-1",                    422
    region: "us-seattle-1"
    region_state: "Production"
    realm: "realm1"
    ...
    service_enclave_dns_suffix: "svc.ad1.r1"},
  { name: "sea-ad-2",                    424
    region: "us-seattle-1"
    region_state: "Production"
    realm: "realm1"
    ...
    service_enclave_dns_suffix: "svc.ad2.r1"},  ]
```

FIG. 4

Flock Config
500

```
504 locals{
506   bootstrap_regions = [for region in local.all_regions : region if region.state
      == 'Building']
508   production_regions = [for region in local.all_regions : region if region.state
      == 'Production' || region.state == 'Paused']
510   region1_bootstrap_regions = [for region in bootstrap_regions : region if
      region.realm == ‘realm1’]
    }                                                                           502

    resource "release_phase" region1_bootstrap {
      name = "region1_bootstrap"
      realm = "region1"
      production = false
    }

    resource "execution_target" region1_bootstrap {
512   count = length(local.region1_bootstrap_regions)
514   name = local.region1_bootstrap_regions[count.index].name

      region = local.region1_bootstrap_regions[count.index]

      phase = release_phase.region1_bootstrap.name
      tenancy_id = "id.tenancy.region1..aaaaaaaaoidp55bcxodh54e5"
      ...
      ...
    }
```

FIG. 5

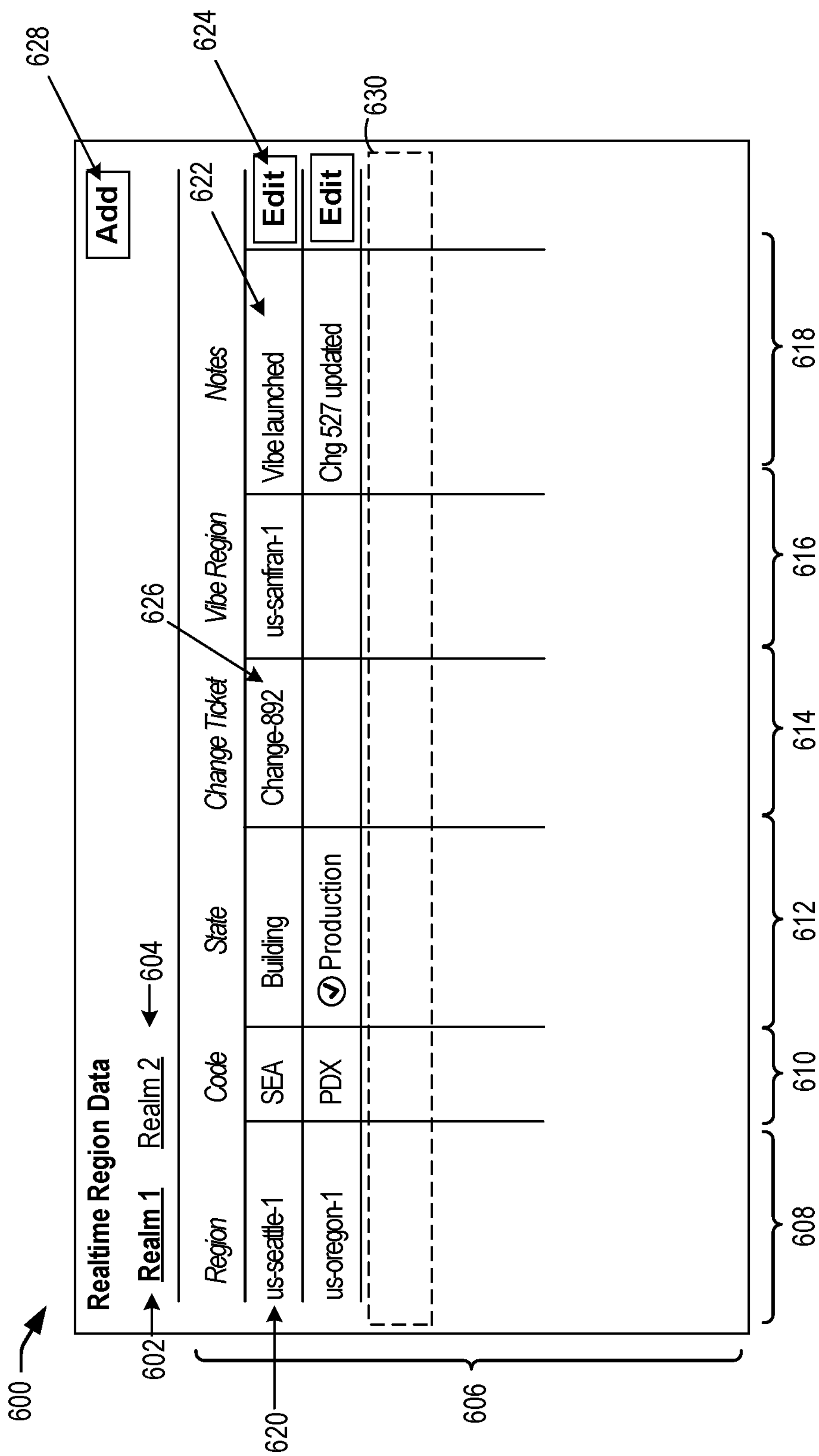

600
602
604
606
608
610
612
614
616
618
620
622
624
626
628
630
Realtime Region Data
Realm 1
Realm 2
Add
Region
Code
State
Change Ticket
Vibe Region
Notes
us-seattle-1
SEA
Building
Change-892
us-sanfran-1
Vibe launched
Edit
us-oregon-1
PDX
Production
Chg 527 updated
Edit

FIG. 6

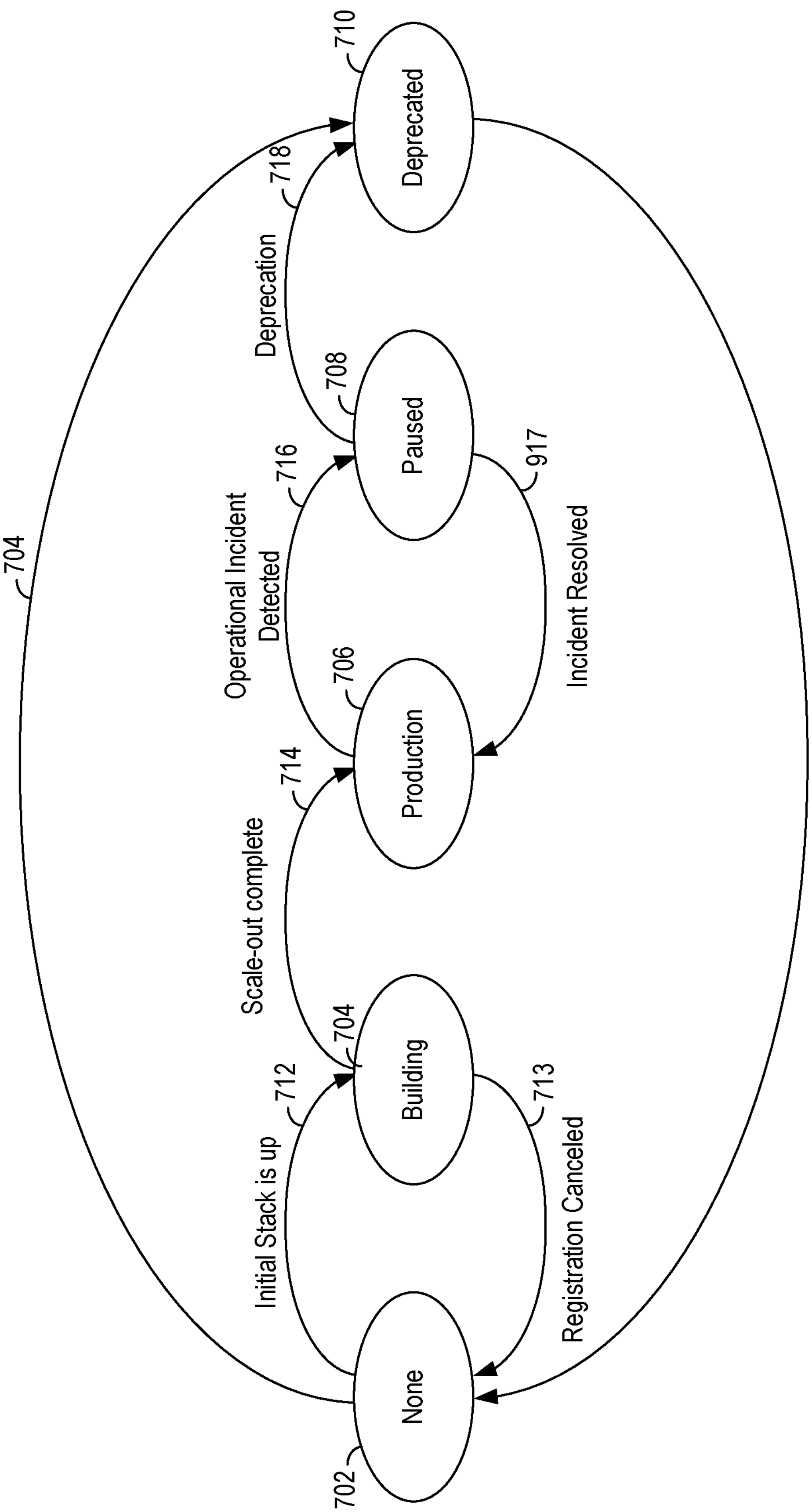

704
710
Deprecated
718
Deprecation
708
Paused
716
Operational Incident
Detected
917
Incident Resolved
706
Production
714
Scale-out complete
704
Building
712
Initial Stack is up
713
Registration Canceled
None
702

FIG. 7

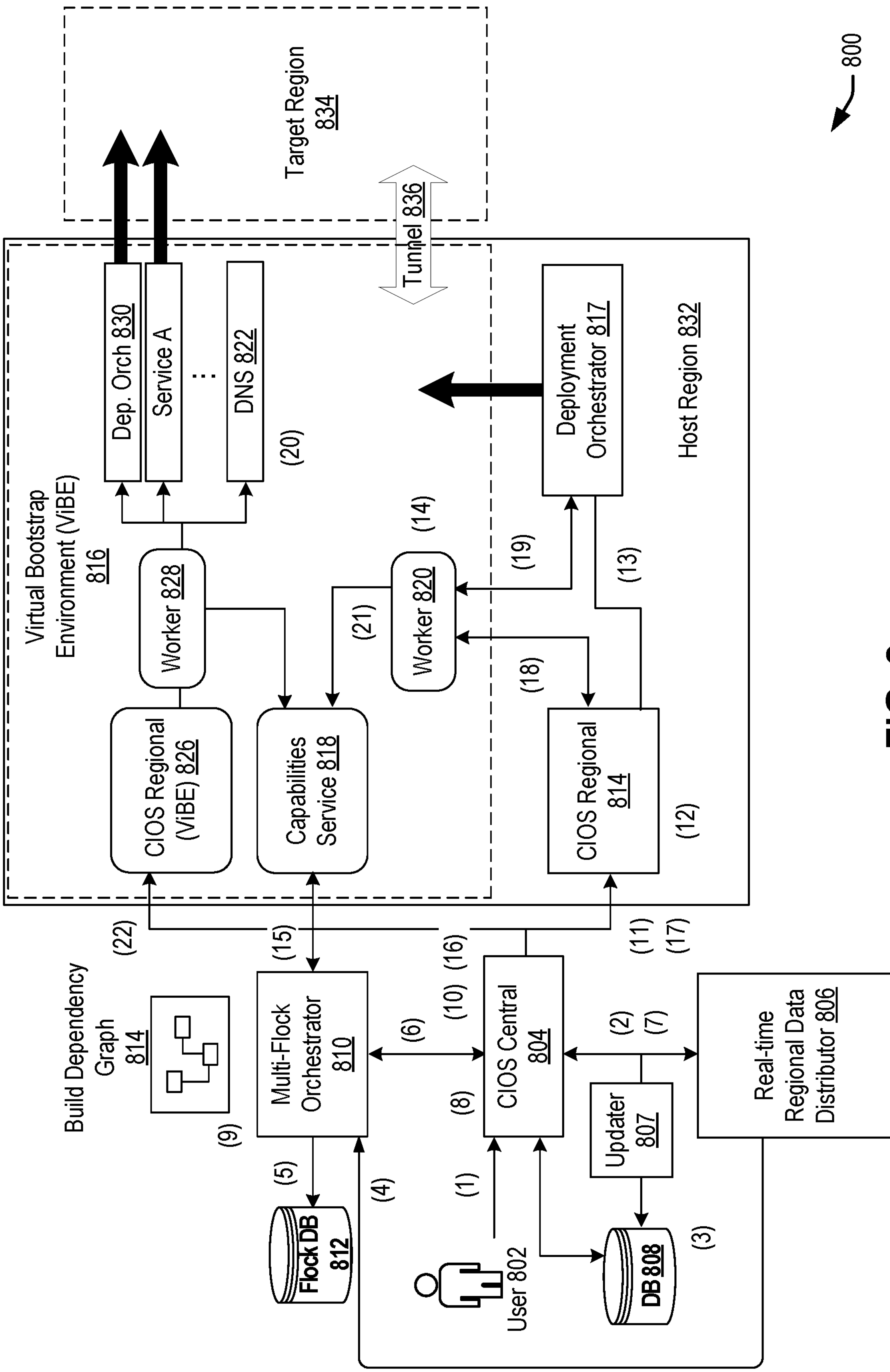

800
Target Region 834
Tunnel 836
Dep. Orch 830
Service A
DNS 822
(20)
Virtual Bootstrap Environment (ViBE) 816
Worker 828
Worker 820
(14)
(19)
(13)
(21)
(18)
Deployment Orchestrator 817
Host Region 832
CIOS Regional (ViBE) 826
Capabilities Service 818
CIOS Regional 814
(12)
(22)
(15)
(16)
(11)
(17)
Build Dependency Graph 814
Multi-Flock Orchestrator 810
(6)
(10)
(8)
CIOS Central 804
(2)
(7)
Updater 807
Real-time Regional Data Distributor 806
(9)
(5)
(4)
(1)
(3)
Flock DB 812
User 802
DB 808

FIG. 8

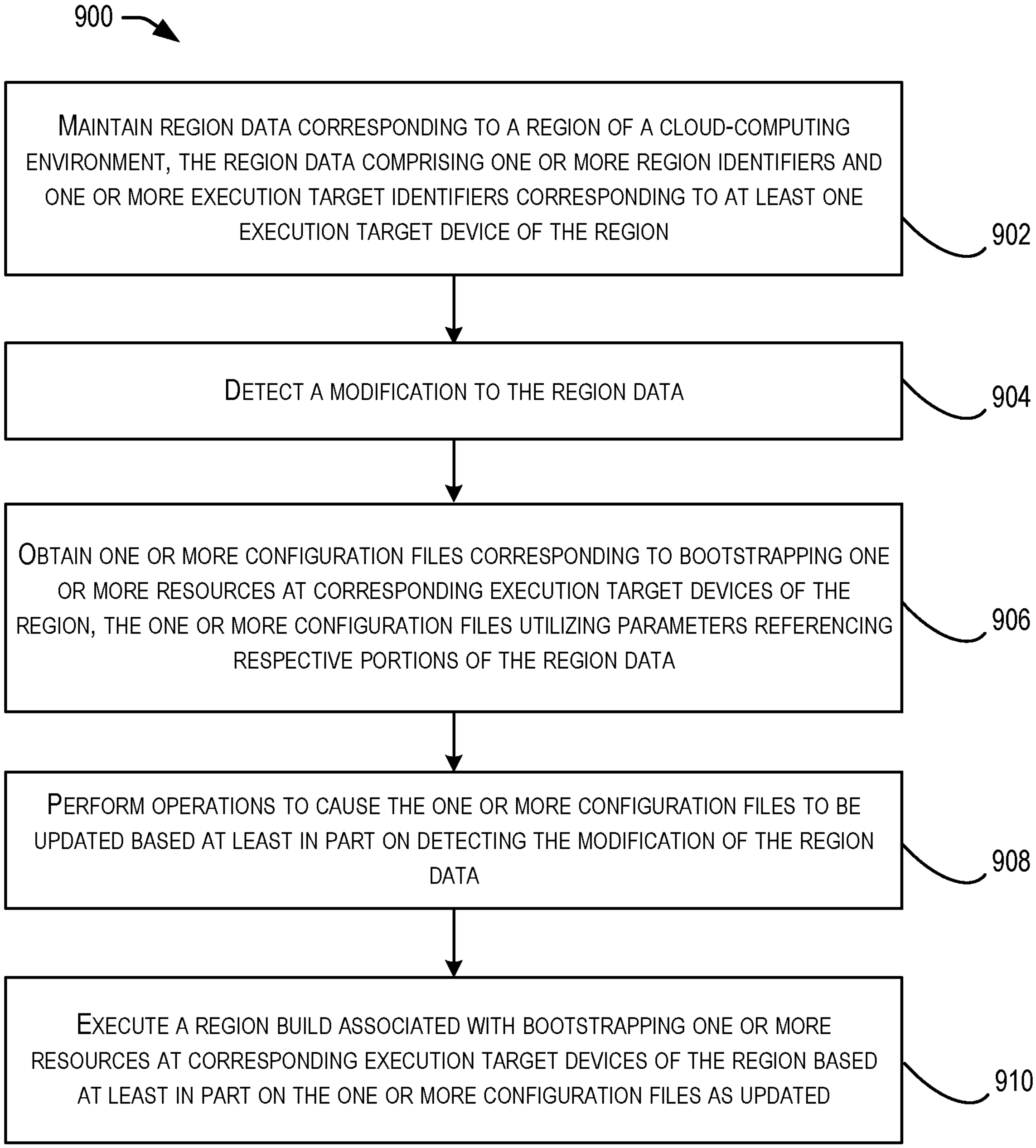

900
MAINTAIN REGION DATA CORRESPONDING TO A REGION OF A CLOUD-COMPUTING ENVIRONMENT, THE REGION DATA COMPRISING ONE OR MORE REGION IDENTIFIERS AND ONE OR MORE EXECUTION TARGET IDENTIFIERS CORRESPONDING TO AT LEAST ONE EXECUTION TARGET DEVICE OF THE REGION
902
DETECT A MODIFICATION TO THE REGION DATA
904
OBTAIN ONE OR MORE CONFIGURATION FILES CORRESPONDING TO BOOTSTRAPPING ONE OR MORE RESOURCES AT CORRESPONDING EXECUTION TARGET DEVICES OF THE REGION, THE ONE OR MORE CONFIGURATION FILES UTILIZING PARAMETERS REFERENCING RESPECTIVE PORTIONS OF THE REGION DATA
906
PERFORM OPERATIONS TO CAUSE THE ONE OR MORE CONFIGURATION FILES TO BE UPDATED BASED AT LEAST IN PART ON DETECTING THE MODIFICATION OF THE REGION DATA
908
EXECUTE A REGION BUILD ASSOCIATED WITH BOOTSTRAPPING ONE OR MORE RESOURCES AT CORRESPONDING EXECUTION TARGET DEVICES OF THE REGION BASED AT LEAST IN PART ON THE ONE OR MORE CONFIGURATION FILES AS UPDATED
910

FIG. 9

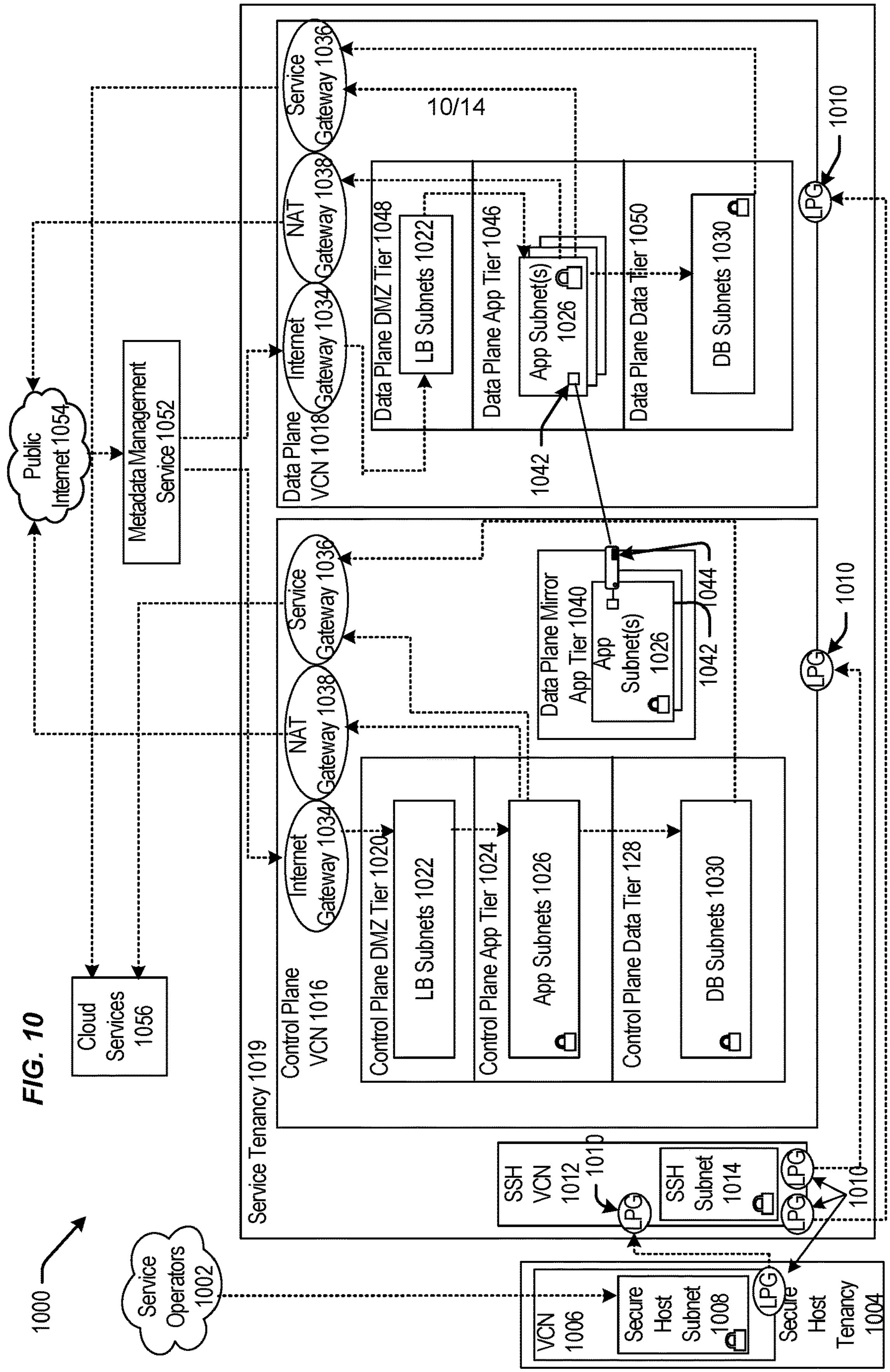

1000
Service Operators 1002
Cloud Services 1056
Public Internet 1054
Metadata Management Service 1052
Service Tenancy 1019
Control Plane VCN 1016
Internet Gateway 1034
NAT Gateway 1038
Service Gateway 1036
Control Plane DMZ Tier 1020
LB Subnets 1022
Control Plane App Tier 1024
App Subnets 1026
Control Plane Data Tier 128
DB Subnets 1030
Data Plane Mirror App Tier 1040
App Subnet(s) 1026
1042
1044
Data Plane VCN 1018
Data Plane DMZ Tier 1048
Data Plane App Tier 1046
Data Plane Data Tier 1050
10/14
LPG
1010
SSH VCN 1012
SSH Subnet 1014
VCN 1006
Secure Host Subnet 1008
Secure Host Tenancy 1004

FIG. 10

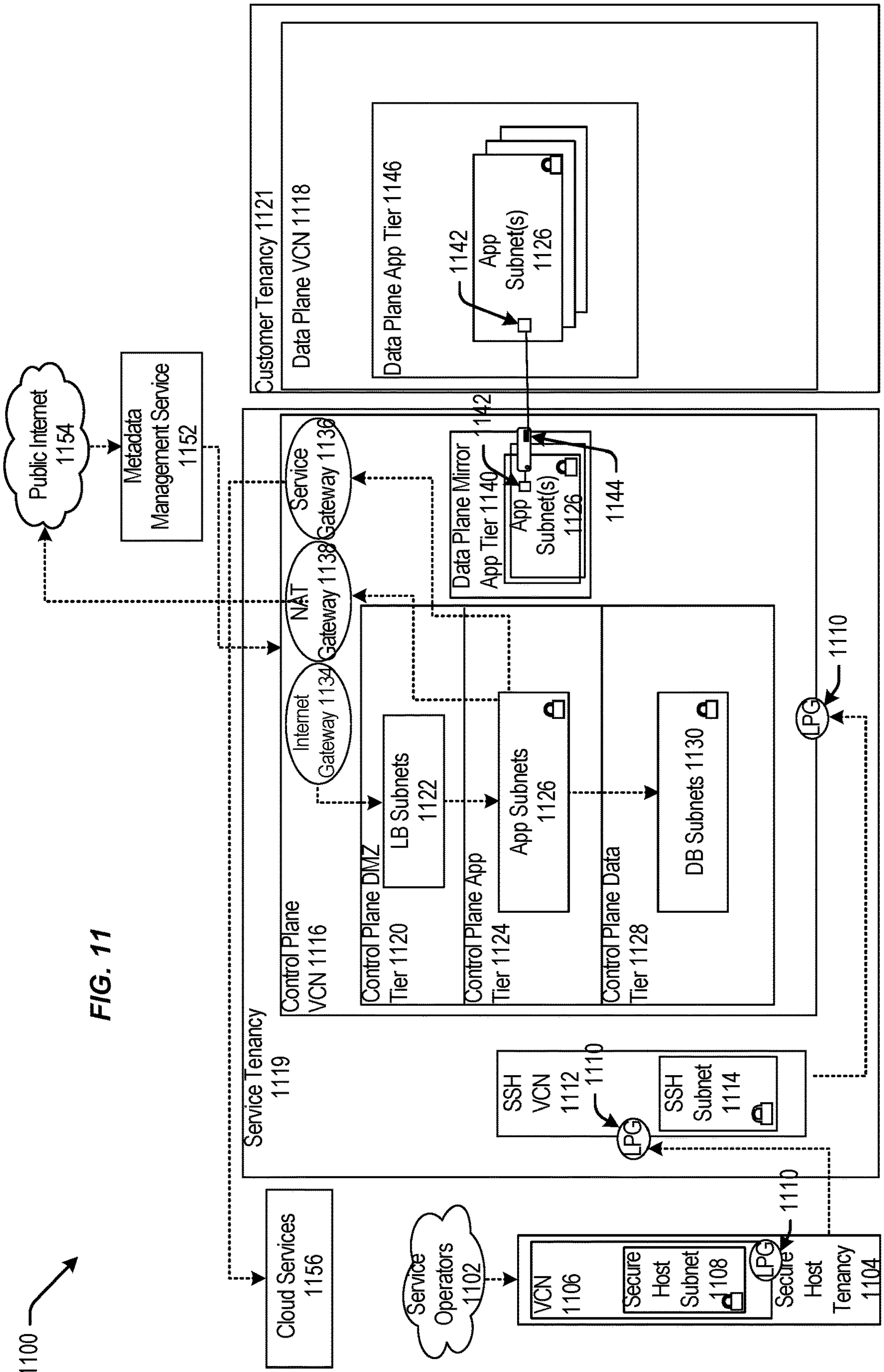

1100
Cloud Services 1156
Service Operators 1102
VCN 1106
Secure Host Subnet 1108
Secure Host Tenancy 1104
LPG
1110
Service Tenancy 1119
SSH VCN 1112
SSH Subnet 1114
Control Plane VCN 1116
Control Plane DMZ Tier 1120
LB Subnets 1122
Control Plane App Tier 1124
App Subnets 1126
Control Plane Data Tier 1128
DB Subnets 1130
Internet Gateway 1134
NAT Gateway 1138
Service Gateway 1136
Data Plane Mirror App Tier 1140
App Subnet(s) 1126
1142
1144
Public Internet 1154
Metadata Management Service 1152
Customer Tenancy 1121
Data Plane VCN 1118
Data Plane App Tier 1146

FIG. 11

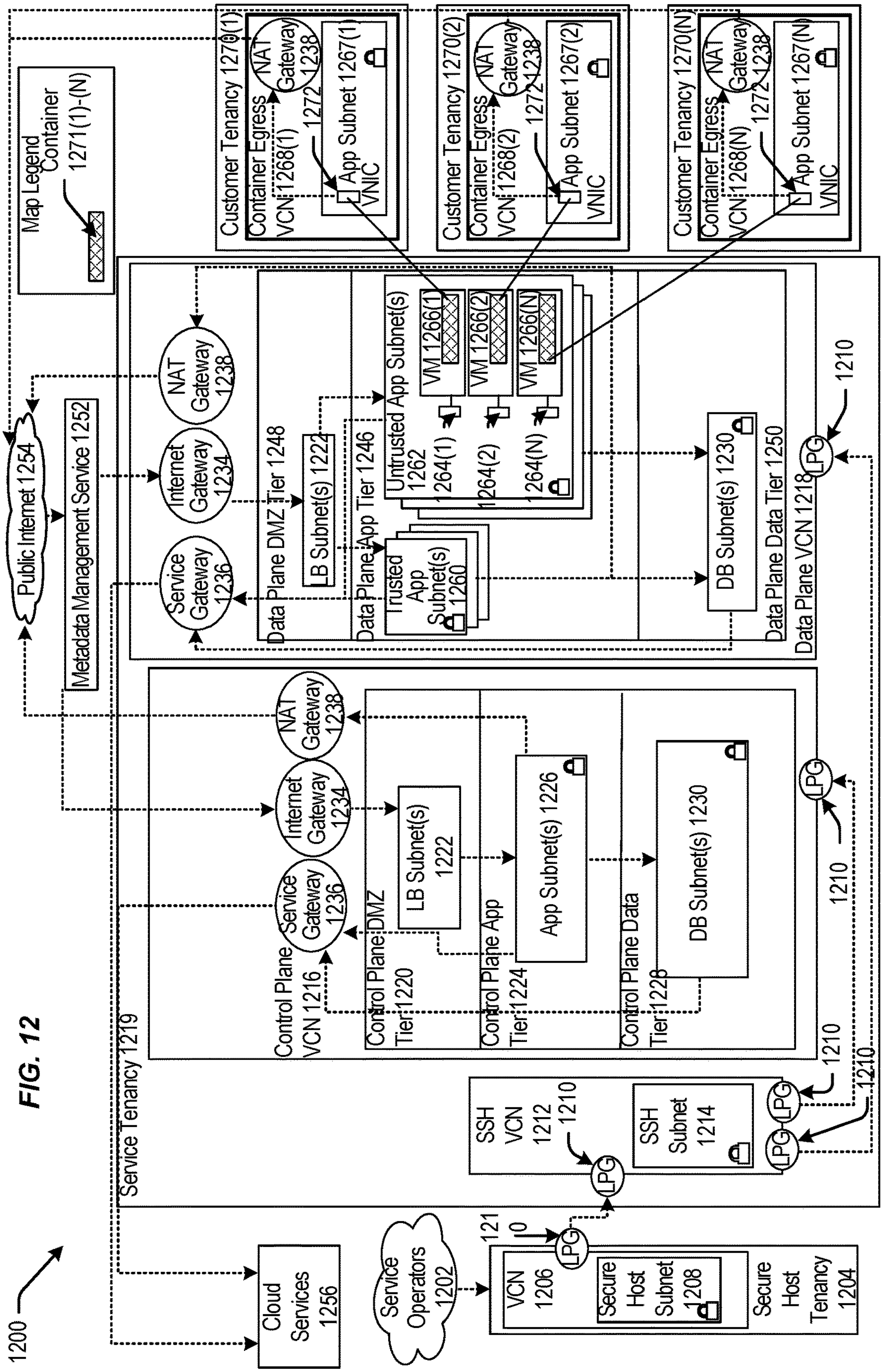

1200
Cloud Services 1256
Service Operators 1202
VCN 1206
Secure Host Subnet 1208
Secure Host Tenancy 1204
LPG
1210
Service Tenancy 1219
SSH VCN 1212
SSH Subnet 1214
Control Plane VCN 1216
Control Plane DMZ Tier 1220
Control Plane App Tier 1224
Control Plane Data Tier 1228
Service Gateway 1236
Internet Gateway 1234
NAT Gateway 1238
LB Subnet(s) 1222
App Subnet(s) 1226
DB Subnet(s) 1230
Metadata Management Service 1252
Public Internet 1254
Data Plane VCN 1218
Data Plane DMZ Tier 1248
Data Plane App Tier 1246
Data Plane Data Tier 1250
Trusted App Subnet(s) 1260
Untrusted App Subnet(s) 1262
1264(1)
1264(2)
1264(N)
VM 1266(1)
VM 1266(2)
VM 1266(N)
Map Legend
Container 1271(1)-(N)
Customer Tenancy 1270(1)
Customer Tenancy 1270(2)
Customer Tenancy 1270(N)
Container Egress VCN 1268(1)
Container Egress VCN 1268(2)
Container Egress VCN 1268(N)
App Subnet 1267(1)
App Subnet 1267(2)
App Subnet 1267(N)
VNIC
1272

FIG. 12

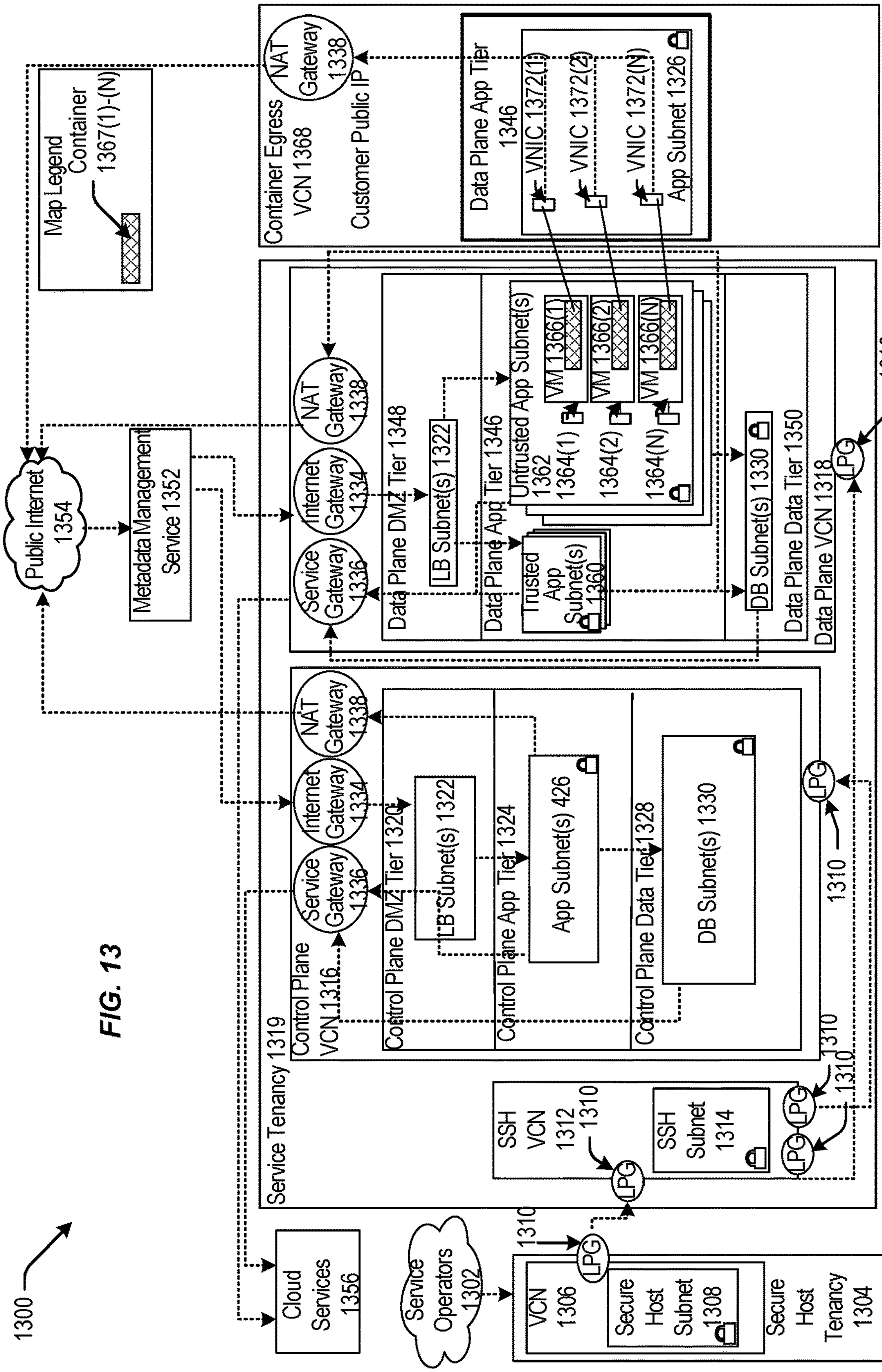

1300
Service Operators 1302
Secure Host Tenancy 1304
VCN 1306
Secure Host Subnet 1308
LPG
1310
Cloud Services 1356
Service Tenancy 1319
SSH VCN 1312
SSH Subnet 1314
Control Plane VCN 1316
Control Plane DMZ Tier 1320
LB Subnet(s) 1322
Control Plane App Tier 1324
App Subnet(s) 426
Control Plane Data Tier 1328
DB Subnet(s) 1330
Service Gateway 1336
Internet Gateway 1334
NAT Gateway 1338
Public Internet 1354
Metadata Management Service 1352
Data Plane VCN 1318
Data Plane DMZ Tier 1348
Data Plane App Tier 1346
Trusted App Subnet(s) 1360
Untrusted App Subnet(s) 1362
1364(1)
1364(2)
1364(N)
VM 1366(1)
VM 1366(2)
VM 1366(N)
Data Plane Data Tier 1350
Container Egress VCN 1368
Customer Public IP
Data Plane App Tier 1346
VNIC 1372(1)
VNIC 1372(2)
VNIC 1372(N)
App Subnet 1326
Map Legend
Container 1367(1)-(N)

FIG. 13

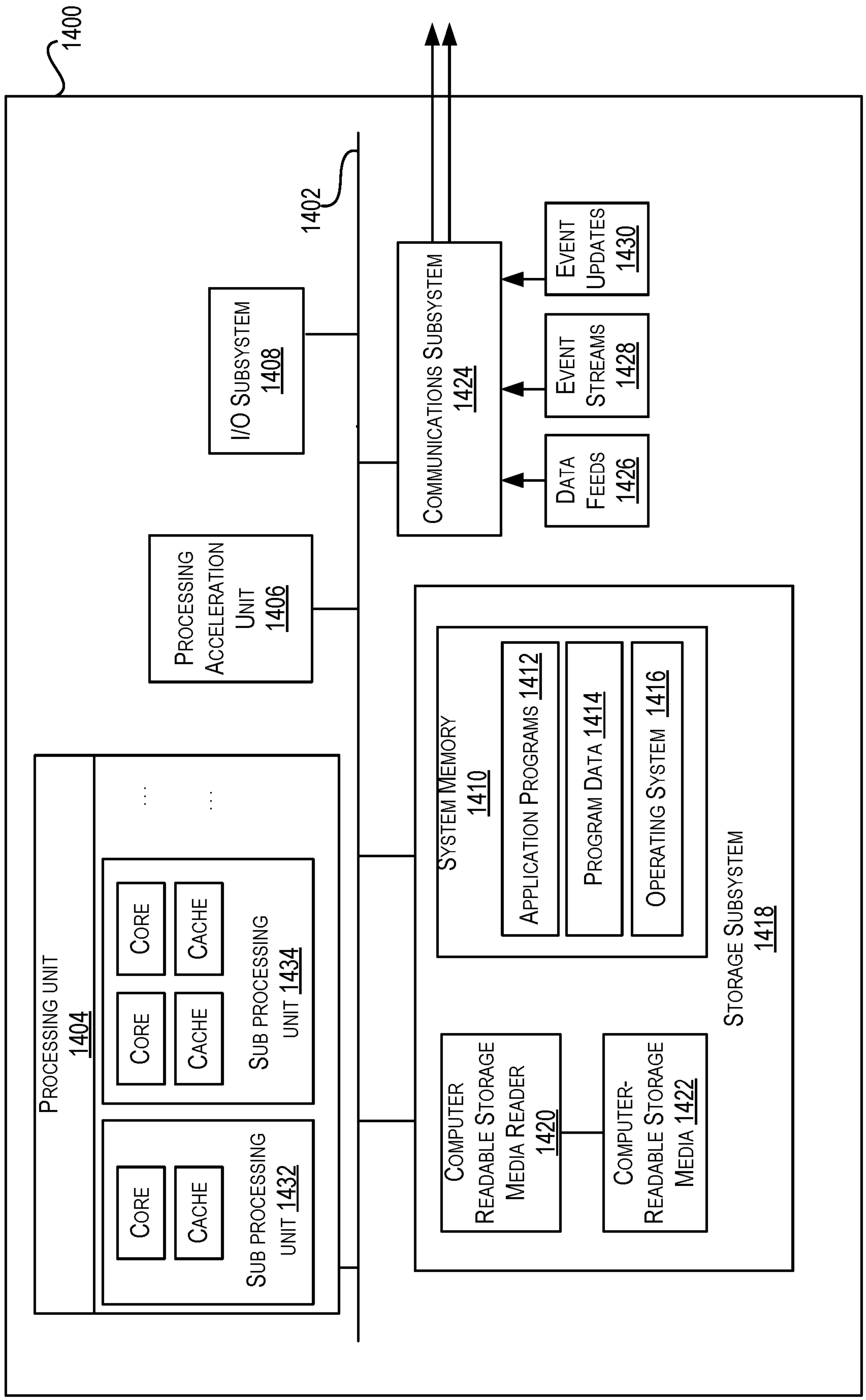

1400
1402
I/O SUBSYSTEM 1408
PROCESSING ACCELERATION UNIT 1406
COMMUNICATIONS SUBSYSTEM 1424
DATA FEEDS 1426
EVENT STREAMS 1428
EVENT UPDATES 1430
PROCESSING UNIT 1404
CORE
CACHE
CORE
CACHE
SUB PROCESSING UNIT 1434
CORE
CACHE
SUB PROCESSING UNIT 1432
SYSTEM MEMORY 1410
APPLICATION PROGRAMS 1412
PROGRAM DATA 1414
OPERATING SYSTEM 1416
STORAGE SUBSYSTEM 1418
COMPUTER READABLE STORAGE MEDIA READER 1420
COMPUTER-READABLE STORAGE MEDIA 1422

FIG. 14

DATA MANAGEMENT TECHNIQUES FOR CLOUD REGIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

This non-provisional application claims priority to U.S. Provisional Patent Application No. 63/308,003, filed on Feb. 8, 2022, entitled "Techniques for Bootstrapping a Region Build," U.S. Provisional Patent Application No. 63/312,814, filed on Feb. 22, 2022, entitled "Techniques for Implementing Virtual Data Centers," and U.S. Provisional Patent Application No. 63/315,024, filed on Feb. 28, 2022, entitled "Data Management Techniques for Cloud Regions," the disclosures of which are herein incorporated by reference in their entirety for all purposes.

BACKGROUND

Today, cloud infrastructure services utilize many individual services to build a data center (e.g., to bootstrap various resources in a data center of a particular geographic region). In some examples, a region is a logical abstraction corresponding to a localized geographical area in which one or more data centers are (or are to be) located. Building a data center may include provisioning and configuring infrastructure resources and deploying code to those resources (e.g., for a variety of services). The operations for building a data center may be collectively referred to as performing a "region build." Any suitable number of data centers may be included in a region and therefore a region build may include operations for building multiple data centers. Conventional tools for building a region require significant manual effort. Additionally, modifying aspects of the region required manual updates, potentially over many files corresponding to various service teams. As the number of service teams and regions grows, the effort required to maintain and modify such data drastically increases. Substantially relying on manual efforts for maintaining region data is time intensive, incurs risks, and may not scale well.

BRIEF SUMMARY

Embodiments of the present disclosure relate to the management and utilization of region data for bootstrapping (provisioning and/or deploying) any suitable number of resources (e.g., infrastructure components and/or software) within one or more data centers of a region (e.g., a geographical location associated with the one or more data centers).

At least one embodiment is directed to a computer-implemented method. The method may include maintaining, by a cloud infrastructure orchestration service, region data corresponding to a region of a cloud-computing environment. In some embodiments, the region data may comprise one or more region identifiers and one or more execution target identifiers corresponding to at least one execution target device of the region. The method may further include detecting, by the cloud infrastructure orchestration service, a modification of the region data. One or more configuration files may be obtained. These configuration files may correspond to bootstrapping one or more resources at corresponding execution target devices of the region. In some embodiments, the one or more configuration files utilizing parameters referencing respective portions of region data. Operations may be performed by the cloud infrastructure orchestration service to cause the one or more configuration files to be updated based at least in part on detecting the modification of the region data. The method may further include executing, cy the cloud infrastructure orchestration service, a region build associated with bootstrapping one or more services with the region based at least in part on the one or more configuration files as updated.

In some embodiments, detecting, by the cloud infrastructure orchestration service, the modification to the region data may further comprise providing one or more user interface for modifying the region data, receiving, at the one or more user interfaces, user input comprising the modification to the region data, and updating the region data in accordance with the user input.

In some embodiments, the method may further comprise maintaining, by a real-time regional data distributor of the cloud infrastructure orchestration service, the region data in a persisted record. The region data may further identify at least one of: an availability domain, an instance of region data corresponding to a virtual bootstrap environment of the cloud infrastructure orchestration service, or a realm identifier.

The method may further comprise maintaining, by the cloud infrastructure orchestration service, a state associated with the region build. The state of the region may be presented at a user interface provided by the cloud infrastructure orchestration service. In some embodiments, the state of the region is maintained as part of the region data.

The operations performed to cause the one or more configuration files to be updated may comprise recompiling the one or more configuration files, thereby injecting the one or more configuration files with the updated region data.

In some embodiments, bootstrapping the one or more services within the region comprises provisioning at least one infrastructure component and deploying at least one artifact to the at least one infrastructure component in accordance with one or more configuration files comprising the region data as updated.

Another embodiment is directed to a computing device hosting a cloud infrastructure orchestration service, the computing device comprising one or more processors and instructions that, when executed by the one or more processors, cause the cloud infrastructure orchestration service to perform the method(s) disclosed herein.

Still another embodiment is directed to a non-transitory computer-readable medium storing computer-executable instructions that, when executed by one or more processors of a cloud infrastructure orchestration service, cause the cloud infrastructure orchestration service to perform the method(s) disclosed herein.

BRIEF DESCRIPTION OF THE DRAWINGS

To easily identify the discussion of any particular element or act, the most significant digit or digits in a reference number refer to the figure number in which that element is first introduced.

FIG. **1** is a block diagram of an environment in which a Cloud Infrastructure Orchestration Service (CIOS) may operate to dynamically provide bootstrap services in a region, according to at least one embodiment.

FIG. **2** is a block diagram for illustrating an environment and method for building a virtual bootstrap environment (ViBE), according to at least one embodiment.

FIG. **3** is a block diagram for illustrating an environment and method for bootstrapping services to a target region utilizing the ViBE, according to at least one embodiment.

FIG. **4** illustrates an example instance of region data, according to at least one embodiment.

FIG. **5** illustrates an example of a configuration file with one or more variables referencing region data, according to at least one embodiment.

FIG. **6** is a block diagram depicting an example user interface for modifying and viewing region data, according to at least one embodiment.

FIG. **7** is a block diagram depicting a method for maintaining region state data, according to at least one embodiment.

FIG. **8** is a block diagram depicting an example method for updating region data, according to at least one embodiment.

FIG. **9** is a block diagram depicting an example method for executing a region build with updated region data, according to at least one embodiment.

FIG. **10** is a block diagram illustrating one pattern for implementing a cloud infrastructure as a service system, according to at least one embodiment.

FIG. **11** is a block diagram illustrating another pattern for implementing a cloud infrastructure as a service system, according to at least one embodiment.

FIG. **12** is a block diagram illustrating another pattern for implementing a cloud infrastructure as a service system, according to at least one embodiment.

FIG. **13** is a block diagram illustrating another pattern for implementing a cloud infrastructure as a service system, according to at least one embodiment.

FIG. **14** is a block diagram illustrating an example computer system, according to at least one embodiment.

DETAILED DESCRIPTION

In the following description, for the purposes of explanation, specific details are set forth in order to provide a thorough understanding of certain embodiments. However, it will be apparent that various embodiments may be practiced without these specific details. The FIGS. and description are not intended to be restrictive. The word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment or design described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments or designs.

Example Automated Data Center Build (Region Build) Infrastructure

The adoption of cloud services has seen a rapid uptick in recent times. Various types of cloud services are now provided by various different cloud service providers (CSPs). The term cloud service is generally used to refer to a service or functionality that is made available by a CSP to users or customers on demand (e.g., via a subscription model) using systems and infrastructure (cloud infrastructure) provided by the CSP. Typically, the servers and systems that make up the CSP's infrastructure, and which are used to provide a cloud service to a customer, are separate from the customer's own on-premises servers and systems. Customers can thus avail themselves of cloud services provided by the CSP without having to purchase separate hardware and software resources for the services. Cloud services are designed to provide a subscribing customer easy, scalable, and on-demand access to applications and computing resources without the customer having to invest in procuring the infrastructure that is used for providing the services or functions. Various different types or models of cloud services may be offered such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS), and others. A customer can subscribe to one or more cloud services provided by a CSP. The customer can be any entity such as an individual, an organization, an enterprise, and the like.

As indicated above, a CSP is responsible for providing the infrastructure and resources that are used for providing cloud services to subscribing customers. The resources provided by the CSP can include both hardware and software resources. These resources can include, for example, compute resources (e.g., virtual machines, containers, applications, processors), memory resources (e.g., databases, data stores), networking resources (e.g., routers, host machines, load balancers), identity, and other resources. In certain implementations, the resources provided by a CSP for providing a set of cloud services CSP are organized into data centers. A data center may be configured to provide a particular set of cloud services. The CSP is responsible for equipping the data center with infrastructure and resources that are used to provide that particular set of cloud services. A CSP may build one or more data centers.

Data centers provided by a CSP may be hosted in different regions. A region is a localized geographic area and may be identified by a region name. Regions are generally independent of each other and can be separated by vast distances, such as across countries or even continents. Regions are grouped into realms. Examples of regions for a CSP may include US West, US East, Australia East, Australia Southeast, and the like.

A region can include one or more data centers, where the data centers are located within a certain geographic area corresponding to the region. As an example, the data centers in a region may be located in a city within that region. For example, for a particular CSP, data centers in the US West region may be located in San Jose, California; data centers in the US East region may be located in Ashburn, Virginia; data centers in the Australia East region may be located in Sydney, Australia; data centers in the Australia Southeast region may be located in Melbourne, Australia; and the like.

Data centers within a region may be organized into one or more availability domains, which are used for high availability and disaster recovery purposes. An availability domain can include one or more data centers within a region. Availability domains within a region are isolated from each other, fault tolerant, and are architected in such a way that data centers in multiple availability domains are very unlikely to fail simultaneously. For example, the availability domains within a region may be structured in a manner such that a failure at one availability domain within the region is unlikely to impact the availability of data centers in other availability domains within the same region.

When a customer or subscriber subscribes to or signs up for one or more services provided by a CSP, the CSP creates a tenancy for the customer. The tenancy is like an account that is created for the customer. In certain implementations, a tenancy for a customer exists in a single realm and can access all regions that belong to that realm. The customer's users can then access the services subscribed to by the customer under this tenancy.

As indicated above, a CSP builds or deploys data centers to provide cloud services to its customers. As a CSP's customer base grows, the CSP typically builds new data centers in new regions or increases the capacity of existing data centers to service the customers' growing demands and to better serve the customers. Preferably, a data center is built in close geographical proximity to the location of customers serviced by that data center. Geographical proximity between a data center and customers serviced by that data center lends to more efficient use of resources and faster and more reliable services being provided to the customers. Accordingly, a CSP typically builds new data centers in new regions in geographical areas that are geographically proximal to the customers serviced by the data centers. For example, for a growing customer base in Germany, a CSP may build one or more data centers in a new region in Germany.

Building a data center (or multiple data centers) in a region is sometimes also referred to as building a region. The term "region build" is used to refer to building one or more data centers in a region. Building a data center in a region involves provisioning or creating a set of new resources that are needed or used for providing a set of services that the data center is configured to provide. The end result of the region build process is the creation of a data center in a region, where the data center is capable of providing a set of services intended for that data center and includes a set of resources that are used to provide the set of services.

Building a new data center in a region is a very complex activity requiring extensive coordination between various bootstrapping activities. At a high level, this involves the performance and coordination of various tasks such as: identifying the set of services to be provided by the data center; identifying various resources that are needed for providing the set of services; creating, provisioning, and deploying the identified resources; wiring the resources properly so that they can be used in an intended manner; and the like. Each of these tasks further have subtasks that need to be coordinated, further adding to the complexity. Due to this complexity, presently, the building of a data center in a region involves several manually initiated or manually controlled tasks that require careful manual coordination. As a result, the task of building a new region (i.e., building one or more data centers in a region) is very time consuming. It can take time, for example many months, to build a data center. Additionally, the process is very error prone, sometimes requiring several iterations before a desired configuration of the data center is achieved, which further adds to the time taken to build a data center. These limitations and problems severely limit a CSP's ability to grow computing resources in a timely manner responsive to increasing customer needs.

The present disclosure describes techniques for reducing build time, reducing computing resource waste, and reducing risk related to building one or more data centers in a region. Instead of weeks and months needed to build a data center in a region in the past, the techniques described herein can be used to build a new data center in a region in a relatively much shorter time, while reducing the risk of errors over conventional approaches.

A Cloud Infrastructure Orchestration Service (CIOS) is disclosed herein that is configured to bootstrap (e.g., provision and deploy) services into a new data center based on predefined configuration files that identify the resources (e.g., infrastructure components and software to be deployed) for implementing a given change to the data center. The CIOS can parse and analyze configuration files (e.g., flock configs) to identify dependencies between resources, execution targets, phases, and flocks. The CIOS may generate specific data structures from the analysis and may use these data structures to drive operations and to manage an order by which services are bootstrapped to a region. The CIOS may utilize these data structures to identify when it can bootstrap a service, when bootstrapping is blocked, and/or when bootstrapping operations associated with a previously blocked service can resume. Advantageously, the CIOS can identify circular dependencies within the data structures and execute operations to eliminate/resolve these circular dependencies prior to task execution. Using these techniques, the CIOS substantially reduces the risk of executing tasks prior to the availability of the resources on which those tasks depend.

Utilizing the techniques disclosed herein, the CIOS may optimize parallel processing to execute changes to a data center while ensuring that tasks are not initiated until the functionality on which those tasks depend is available in the region. In this manner, the CIOS enables a region build to be performed more efficiently, which greatly reduces the time required to build a data center and the wasteful computing resource use found in conventional approaches.

The present disclosure is directed to maintaining and utilizing region data for use in a region build (e.g., provisioning and/or deploying to one or more data centers of a given region). Conventionally, service teams were required to manually update and identify various region data within their respective flock configuration files (referred to herein as "flock configs" for brevity). Various changes to the region (e.g., adding a new execution target, etc.) required manual updates by various individuals responsible for maintaining those respective configuration files. There was also delay in notifying those service teams of the change to the region. This produced delay in performing the region build and invited a degree of risk that errors would inadvertently be introduced into those configuration files. Utilizing the techniques discussed herein enables region data to be viewed and modified in real time. The modified region data can be applied throughout the system to update any suitable aspect of the region and the resources bootstrapped within the region. These techniques drastically reduce or eliminate the required manual efforts of conventional systems, decrease the time between a change in a region and completion of a corresponding region build, and greatly reduces the risk of inadvertent human error.

Certain Definitions

A "region" is a logical abstraction corresponding to a geographical location. A region can include any suitable number of one or more execution targets. In some embodiments, an execution target could correspond to a data center.

An "execution target" refers to a smallest unit of change for executing a release. A "release" refers to a representation of an intent to orchestrate a specific change to a service (e.g., deploy version 8, "add an internal DNS record," etc.). For most services, an execution target represents an "instance" of a service. A single service can be bootstrapped to each of one or more execution targets. An execution target may be associated with a set of devices (e.g., a data center).

"Bootstrapping" is intended to refer to the collective tasks associated with provisioning and deployment of any suitable number of resources (e.g., infrastructure components, artifacts, etc.) corresponding to a single service.

A "service" refers to functionality provided by a set of resources. A set of resources for a service includes any suitable combination of infrastructure, platform, or software (e.g., an application) hosted by a cloud provider that can be configured to provide the functionality of a service. A service can be made available to users through the Internet.

An "artifact" refers to code being deployed to an infrastructure component or a Kubernetes engine cluster, this may include software (e.g., an application), configuration information (e.g., a configuration file) for an infrastructure component, or the like.

A "flock config" refers to a configuration file (or a set of configuration files) that describes a set of all resources (e.g., infrastructure components and artifacts) associated with a single service. A flock config may include declarative statements that specify one or more aspects corresponding to a desired state of the resources of the service.

"Service state" refers to a point-in-time snapshot of every resource (e.g., infrastructure resources, artifacts, etc.) associated with the service. The service state indicates status corresponding to provisioning and/or deployment tasks associated with service resources.

IaaS provisioning (or "provisioning") refers to acquiring computers or virtual hosts for use, and even installing needed libraries or services on them. The phrase "provisioning a device" refers to evolving a device to a state in which it can be utilized by an end-user for their specific use. A device that has undergone the provisioning process may be referred to as a "provisioned device." Preparing the provisioned device (installing libraries and daemons) may be part of provisioning; this preparation is different from deploying new applications or new versions of an application onto the prepared device. In most cases, deployment does not include provisioning, and the provisioning may need to be performed first. Once prepared, the device may be referred to as "an infrastructure component."

IaaS deployment (or "deployment") refers to the process of providing and/or installing a new application, or a new version of an application, onto a provisioned infrastructure component. Once the infrastructure component has been provisioned (e.g., acquired, assigned, prepared, etc.), additional software may be deployed (e.g., provided to and installed on the infrastructure component). The infrastructure component can be referred to as a "resource" after provisioning and deployment has concluded. Examples of resources may include, but are not limited to, virtual machines, databases, object storage, block storage, load balancers, and the like.

A "capability" identifies a unit of functionality associated with a service. The unit could be a portion, or all, of the functionality to be provided by the service. By way of example, a capability can be published indicating that a resource is available for authorization/authentication processing (e.g., a subset of the functionality to be provided by the resource). As another example, a capability can be published indicating the full functionality of the service is available. Capabilities can be used to identify functionality on which a resource or service depends and/or functionality of a resource or service that is available for use.

A "virtual bootstrap environment" (ViBE) refers to a virtual cloud network that is provisioned in the overlay of an existing region (e.g., a "host region"). Once provisioned, a ViBE is connected to a new region using a communication channel (e.g., an IPsec Tunnel VPN). Certain essential core services (or "seed" services) like a deployment orchestrator, a public key infrastructure (PKI) service, and the like can be provisioned in a ViBE. These services can provide the capabilities required to bring the hardware online, establish a chain of trust to the new region, and deploy the remaining services in the new region. Utilizing the virtual bootstrap environment can prevent circular dependencies between bootstrapping resources by utilizing resources of the host region. Services can be staged and tested in the ViBE prior to the physical region (e.g., the target region) being available.

A "Cloud Infrastructure Orchestration Service" (CIOS) may refer to a system configured to manage provisioning and deployment operations for any suitable number of services as part of a region build.

A Multi-Flock Orchestrator (MFO) may be a computing component (e.g., a service) that coordinates events between components of the CIOS to provision and deploy services to a target region (e.g., a new region). An MFO tracks relevant events for each service of the region build and takes actions in response to those events.

A "host region" refers to a region that hosts a virtual bootstrap environment (ViBE). A host region may be used to bootstrap a ViBE.

A "target region" refers to a region under build.

"Publishing a capability" refers to "publishing" as used in a "publisher-subscriber" computing design or otherwise providing an indication that a particular capability is available (or unavailable). The capabilities are "published" (e.g., collected by a capabilities service, provided to a capabilities service, pushed, pulled, etc.) to provide an indication that functionality of a resource/service is available. In some embodiments, capabilities may be published/transmitted via an event, a notification, a data transmission, a function call, an API call, or the like. An event (or other notification/data transmission/etc.) indicating availability of a particular capability can be broadcasted/addressed (e.g., published) to a capabilities service.

A "Capabilities Service" may be a flock configured to model dependencies between different flocks. A capabilities service may be provided within a Cloud Infrastructure Orchestration Service and may define what capabilities, services, features have been made available in a region.

A "Real-time Regional Data Distributor" (RRDD) may be a service or system configured to manage region data. This region data can be injected into flock configs to dynamically create execution targets for new regions.

In some examples, techniques for implementing a Cloud Infrastructure Orchestration Service (CIOS) are described herein. Such techniques, as described briefly above, can be configured to manage bootstrapping (e.g., provisioning and deploying software to) infrastructure components within a cloud environment (e.g., a region). In some instances, the CIOS can include computing components (e.g., a CIOS Central and a CIOS Regional, both of which will be described in further detail below) that may be configured to manage bootstrapping tasks (provisioning and deployment) for a given service and a Multi-Flock Orchestrator (also described in further detail below) configured to initiate/manage region builds (e.g., bootstrapping operations corresponding to multiple services).

The CIOS enables region building and world-wide infrastructure provisioning and code deployment with minimal manual run-time effort from service teams (e.g., beyond an initial approval and/or physical transportation of hardware, in some instances). The high-level responsibilities of the CIOS include, but are not limited to, coordinating region builds, providing users with a view of the current state of resources managed by the CIOS (e.g., of a region, across regions, world-wide, etc.), and managing bootstrapping operations for bootstrapping resources within a region.

The CIOS may provide view reconciliation, where a view of a desired state (e.g., a desired configuration) of resources may be reconciled with a current/actual state (e.g., a current configuration) of the resources. In some instances, view reconciliation may include obtaining state data to identify what resources are actually running and their current configuration and/or state. Reconciliation can be performed at a variety of granularities, such as at a service level.

The CIOS can perform plan generation, where differences between the desired and current state of the resources are identified. Part of plan generation can include identifying the operations that would need to be executed to bring the resources from the current state to the desired state. In some examples, the CIOS may present a generated plan to a user for approval. In these examples, the CIOS can mark the plan as approved or rejected based on user input from the user. Thus, users can spend less time reasoning about the plan and the plans are more accurate because they are machine generated. Plans are almost too detailed for human consumption; however, the CIOS can provide this data via a sophisticated user interface (UI).

In some examples, the CIOS can handle execution of change management by executing the approved plan. Once an execution plan has been created and approved, engineers may no longer need to participate in change management unless the CIOS initiates roll-back. The CIOS can handle rolling back to a previous service version by generating a plan that returns the service to a previous (e.g., pre-release) state (e.g., when CIOS detects service health degradation while executing).

The CIOS can measure service health by monitoring alarms and executing integration tests. The CIOS can help teams quickly define roll-back behavior in the event of service degradation, which it can later execute. The CIOS can generate and display plans and can track approval. The CIOS can combine the functionality of provisioning and deployment in a single system that coordinates these tasks across a region build. The CIOS also supports the discovery of flocks (e.g., service resources such as flock config(s) corresponding to any suitable number of services), artifacts, resources, and dependencies. The CIOS can discover dependencies between execution tasks at every level (e.g., resource level, execution target level, phase level, service level, etc.) through a static analysis (e.g., including parsing and processing content) of one or more configuration files. Using these dependencies, the CIOS can generate various data structures from these dependencies that can be used to drive task execution (e.g., tasks regarding provisioning of infrastructure resources and deployment of artifacts across the region).

FIG. **1** is a block diagram of an environment **100** in which a Cloud Infrastructure Orchestration Service (CIOS) **102** may operate to dynamically provide bootstrap services in a region, according to at least one embodiment. CIOS **102** can include, but is not limited to, the following components: Real-time Regional Data Distributor (RRDD) **104**, Multi-Flock Orchestrator (MFO) **106**, CIOS Central **108**, CIOS Regional **110**, and Capabilities Service **112**. Specific functionality of CIOS Central **108** and CIOS Regional **110** is provided in more detail in U.S. application Ser. No. 17/016,754, entitled "Techniques for Deploying Infrastructure Resources with a Declarative Provisioning Tool," the entire contents of which are incorporated in its entirety for all purposes. In some embodiments, any suitable combination of the components of CIOS **102** may be provided as a service. In some embodiments, some portion of CIOS **102** may be deployed to a region (e.g., a data center represented by host region **103**). In some embodiments, CIOS **102** may include any suitable number of cloud services (not depicted in FIG. **1**) discussed in further detail in U.S. application Ser. No. 17/016,754 and below with respect to FIGS. **2** and **3**.

Real-time Regional Data Distributor (RRDD) **104** may be configured to maintain and provide region data that identifies realms, regions, execution targets, and availability domains. In some cases, the region data may be in any suitable form (e.g., JSON format, data objects/containers, XML, etc.). Region data maintained by RRDD **104** may include any suitable number of subsets of data which can individually be referenceable by a corresponding identifier. By way of example, an identifier "all_regions" can be associated with a data structure (e.g., a list, a structure, an object, etc.) that includes a metadata for all defined regions. As another example, an identifier such as "realms" can be associated with a data structure that identifies metadata for a number of realms and a set of regions corresponding to each realm. In general, the region data may maintain any suitable attribute of one or more realm(s), region(s), availability domains (ADs), execution target(s) (ETs), and the like, such as identifiers, DNS suffixes, states (e.g., a state of a region), and the like. The RRDD **104** may be configured to manage region state as part of the region data. A region state may include any suitable information indicating a state of bootstrapping within a region. By way of example, some example region states can include "initial," "building," "production," "paused," or "deprecated." The "initial" state may indicate a region that has not yet been bootstrapped. A "building" state may indicate that bootstrapping of one or more flocks within the region has commenced. A "production" state may indicate that bootstrapping has been completed and the region is ready for validation. A "paused" state may indicate that CIOS Central **108** or CIOS Regional **110** has paused internal interactions with the regional stack, likely due to an operational issue. A "deprecated" state may indicate the region has been deprecated and is likely unavailable and/or will not be contacted again.

CIOS Central **108** is configured to provide any suitable number of user interfaces with which users (e.g., user **109**) may interact with CIOS **102**. By way of example, users can make changes to region data via a user interface provided by CIOS Central **108**. CIOS Central **108** may additionally provide a variety of interfaces that enable users to: view changes made to flock configs and/or artifacts, generate and view plans, approve/reject plans, view status on plan execution (e.g., corresponding to tasks involving infrastructure provisioning, deployment, region build, and/or desired state of any suitable number of resources managed by CIOS **102**. CIOS Central **108** may implement a control plane configured to manage any suitable number of CIOS Regional **110** instances. CIOS Central **108** can provide one or more user interfaces for presenting region data, enabling the user **109** to view and/or change region data. CIOS Central **108** can be configured to invoke the functionality of RRDD **104** via any suitable number of interfaces. Generally, CIOS Central **108** may be configured to manage region data, either directly or indirectly (e.g., via RRDD **104**). CIOS Central **108** may be configured to compile flock configs to inject region data as variables within the flock configs.

Each instance of CIOS Regional **110** may correspond to a module configured to execute bootstrapping tasks that are associated with a single service of a region. CIOS Regional **110** can receive desired state data from CIOS Central **108**. In some embodiments, desired state data may include a flock config that declares (e.g., via declarative statements) a desired state of resources associated with a service. CIOS Central **108** can maintain current state data indicating any suitable aspect of the current state of the resources associated with a service. In some embodiments, CIOS Regional **110** can identify, through a comparison of the desired state data and the current state data, that changes are needed to one or more resources. For example, CIOS Regional **110** can determine that one or more infrastructure components need to be provisioned, one or more artifacts deployed, or any suitable change needed to the resources of the service to bring the state of those resources in line with the desired state. As CIOS Regional **110** performs bootstrapping operations, it may publish data indicating various capabilities of a resource as they become available. A "capability" identifies a unit of functionality associated with a service. The unit could be a portion, or all of the functionality to be provided by the service. By way of example, a capability can be published indicating that a resource is available for authorization/authentication processing (e.g., a subset of the functionality to be provided by the resource). As another example, a capability can be published indicating the full functionality of the service is available. Capabilities can be used to identify functionality on which a resource or service depends and/or functionality of a resource or service that is available for use.

Capabilities Service **112** is configured to maintain capabilities data that indicates 1) what capabilities of various services are currently available, 2) whether any resource/service is waiting on a particular capability, 3) what particular resources and/or services are waiting on a given capability, or any suitable combination of the above. Capabilities Service **112** may provide an interface with which capabilities data may be requested. Capabilities Service **112** may provide one or more interfaces (e.g., application programming interfaces) that enable it to transmit capabilities data to MFO **106** and/or CIOS Regional **110** (e.g., each instance of CIOS Regional **110**). In some embodiments, MFO **106** and/or any suitable component or module of CIOS Regional **110** may be configured to request capabilities data from Capabilities Service **112**.

In some embodiments, Multi-Flock Orchestrator (MFO) **106** may be configured to drive region build efforts. In some embodiments, MFO **106** can manage information that describes what flock/flock config versions and/or artifact versions are to be utilized to bootstrap a given service within a region (or to make a unit of change to a target region). In some embodiments, MFO **106** may be configured to monitor (or be otherwise notified of) changes to the region data managed by Real-time Regional Data Distributor **104**. In some embodiments, receiving an indication that region data has been changed may cause a region build to be triggered by MFO **106**. In some embodiments, MFO **106** may collect various flock configs and artifacts to be used for a region build. Some, or all, of the flock configs may be configured to be region agnostic. That is, the flock configs may not explicitly identify what regions to which the flock is to be bootstrapped. In some embodiments, MFO **106** may trigger a data injection process through which the collected flock configs are recompiled (e.g., by CIOS Central **108**). During recompilation, operations may be executed (e.g., by CIOS Central **108**) to cause the region data maintained by Real-time Regional Data Distributor **104** to be injected into the config files. Flock configs can reference region data through variables/parameters without requiring hard-coded identification of region data. The flock configs can be dynamically modified at run time using this data injection rather than having the region data be hardcoded, and therefore, and more difficult to change.

Multi-Flock Orchestrator **106** can perform a static flock analysis in which the flock configs are parsed to identify dependencies between resources, execution targets, phases, and flocks, and in particular to identify circular dependencies that need to be removed. In some embodiments, MFO **106** can generate any suitable number of data structures based on the dependencies identified. These data structures (e.g., directed acyclic graph(s), linked lists, etc.) may be utilized by the Cloud Infrastructure Orchestration Service **102** to drive operations for performing a region build. By way of example, these data structures may collectively define an order by which services are bootstrapped within a region. An example of such a data structure is discussed further below with respect to Build Dependency Graph **338** of FIG. **3**. If circular dependencies (e.g., service A requires service B and vice versa) exist and are identified through the static flock analysis and/or graph, MFO may be configured to notify any suitable service teams that changes are required to the corresponding flock config to correct these circular dependencies. MFO **106** can be configured to traverse one or more data structures to manage an order by which services are bootstrapped to a region. MFO **106** can identify (e.g., using data obtained from Capabilities Service **112**) capabilities available within a given region at any given time. MFO **106** can this data to identify when it can bootstrap a service, when bootstrapping is blocked, and/or when bootstrapping operations associated with a previously blocked service can resume. Based on this traversal, MFO **106** can perform a variety of releases in which instructions are transmitted by MFO **106** to CIOS Central **108** to perform bootstrapping operations corresponding to any suitable number of flock configs. In some examples, MFO **106** may be configured to identify that one or more flock configs may require multiple releases due to circular dependencies found within the graph. As a result, MFO **106** may transmit multiple instruction sets to CIOS Central **108** for a given flock config to break the circular dependencies identified in the graph.

In some embodiments, a user can request that a new region (e.g., target region **114**) be built. This can involve bootstrapping resources corresponding to a variety of services. In some embodiments, target region **114** may not be communicatively available (and/or secure) at a time at which the region build request is initiated. Rather than delay bootstrapping until such time as target region **114** is available and configured to perform bootstrapping operations, CIOS **102** may initiate the region build using a virtual bootstrap environment **116**. Virtual bootstrap environment (ViBE) **116** may be an overlay network that is hosted by host region **103** (a preexisting region that has previously been configured with a core set of services and which is communicatively available and secure). MFO **106** can leverage resources of the host region **103** to bootstrap resources to the ViBE **116** (generally referred to as "building the ViBE"). By way of example, MFO **106** can provide instructions through CIOS Central **108** that cause an instance of CIOS Regional **110** within a host region (e.g., host region **103**) to bootstrap another instance of CIOS Regional within the ViBE **116**. Once the CIOS Regional within the ViBE is available for processing, bootstrapping the services for the target region **114** can continue within the ViBE **116**. When target region **114** is available to perform bootstrapping operations, the previously bootstrapped services within ViBE **116** may be migrated to target region **114**. Utilizing these techniques, CIOS **102** can greatly improve the speed at which a region is built by drastically reducing the need for any manual input and/or configuration to be provided.

FIG. **2** is a block diagram for illustrating an environment **200** and method for building a virtual bootstrap environment (ViBE) **202** (an example of ViBE **116** of FIG. **1**), according to at least one embodiment. ViBE **202** represents a virtual cloud network that is provisioned in the overlay of an existing region (e.g., host region **204**, an example of the host region **103** of FIG. **1** and in an embodiment is a Host Region Service Enclave). ViBE **202** represents an environment in which services can be staged for a target region (e.g., a region under build such as target region **114** of FIG. **1**) before the target region becomes available.

In order to bootstrap a new region (e.g., target region **114** of FIG. **1**), a core set of services may be bootstrapped. While those core set of services exist in the host region **204**, they do not yet exist in the ViBE (nor the target region). These essential core services provide the functionality needed to provision devices, establish a chain of trust to the new region, and deploy remaining services (e.g., flocks) into a region. The ViBE **202** may be a tenancy that is deployed in a host region **204**. It can be thought of as a virtual region.

When the target region is available to provide bootstrapping operations, the ViBE **202** can be connected to the target region so that services in the ViBE can interact with the services and/or infrastructure components of the target region. This will enable deployment of production level services, instead of self-contained seed services as in previous systems, and will require connectivity over the internet to the target region. Conventionally, a seed service was deployed as part of a container collection and used to bootstrap dependencies necessary to build out the region. Using infrastructure/tooling of an existing region, resources may be bootstrapped (e.g., provisioned and deployed) into the ViBE **202** and connected to the service enclave of a region (e.g., host region **204**) in order to provision hardware and deploy services until the target region is self-sufficient and can be communicated with directly. Utilizing the ViBE **202** allows for standing up the dependencies and services needed to be able to provision/prepare infrastructure and deploy software while making use of the host region's resources in order to break circular dependencies of core services.

Multi-Flock Orchestrator (MFO) **206** may be configured to perform operations to build (e.g., configure) ViBE **202**. MFO **206** can obtain applicable flock configs corresponding to various resources to be bootstrapped to the new region (in this case, a ViBE region, ViBE **202**). By way of example, MFO **206** may obtain a flock config (e.g., a "ViBE flock config") that identifies aspects of bootstrapping Capabilities Service **208** and Worker **210**. As another example, MFO **206** may obtain another flock config corresponding to bootstrapping Domain Name Service (DNS) **212** to ViBE **202**.

At step 1, MFO **206** may instruct CIOS Central **214** (e.g., an example of CIOS Central **108** and CIOS Central **214** of FIGS. **1** and **2**, respectively). For example, MFO **206** may transmit a request (e.g., including the ViBE flock config) to request bootstrapping of the Capabilities Service **208** and Worker **210** that, at this time do not yet exist in the ViBE **202**. In some embodiments, CIOS Central **214** may have access to all flock configs. Therefore, in some examples, MFO **206** may transmit an identifier for the ViBE flock config rather than the file itself, and CIOS Central **214** may independently obtain it from storage (e.g., from DB **308** or flock DB **312** of FIG. **3**).

At step 2, CIOS Central **214** may provide the ViBE flock config via a corresponding request to CIOS Regional **216**. CIOS Regional **216** may parse the ViBE flock config to identify and execute specific infrastructure provisioning and deployment operations at step 3.

In some embodiments, the CIOS Regional **216** may utilize additional corresponding services for provisioning and deployment. For example, at step 4, CIOS Regional **216** CIOS Regional may instruct deployment orchestrator **218** (e.g., an example of a core service, or other write, build, and deploy applications software, of the host region **204**) to execute instructions that in turn cause Capabilities Service **208** and Worker **210** to be bootstrapped within ViBE **202**.

At step 5, a capability may be transmitted to the Capabilities Service **208** (from the CIOS Regional **216**, Deployment Orchestrator **218** via the Worker **210** or otherwise) indicating that resources corresponding to the ViBE flock are available. Capabilities Service **208** may persist this data. In some embodiments, the Capabilities Service **208** adds this information to a list it maintains of available capabilities with the ViBE. By way of example, the capability provided to Capabilities Service **208** at step 5 may indicate the Capabilities Service **208** and Worker **210** are available for processing.

At step 6, MFO **206** may identify that the capability indicating that Capabilities Service **208** and Worker **210** are available based on receiving or obtaining data (an identifier corresponding to the capability) from the Capabilities Service **208**.

At step 7, as a result of receiving/obtaining the data at step 6, the MFO **206** may instruct CIOS Central **214** to bootstrap a DNS service (e.g., DNS **212**) to the ViBE **202**. The instructions may identify or include a particular flock config corresponding to the DNS service.

At step 8, the CIOS Central **214** may instruct the CIOS Regional **216** to deploy DNS **212** to the ViBE **202**. In some embodiments, the DNS flock config for the DNS **212** is provided by the CIOS Central **214**.

At step 9, Worker **210**, now that it is deployed in the ViBE **202**, may be assigned by CIOS Regional **216** to the task of deploying DNS **212**. Worker may execute a declarative infrastructure provisioner in the manner described above in connection with FIG. **3** to identify (e.g., from comparing the flock config (the desired state) to a current state of the (currently non-existing) resources associated with the flock) a set of operations that need to be executed to deploy DNS **212**.

At step 10, the Deployment Orchestrator **218** may instruct Worker **210** to deploy DNS **212** in accordance with the operations identified at step 9. As depicted, Worker **210** proceeds with executing operations to deploy DNS **212** to ViBE **202** at step 11. At step 12, Worker **210** notifies Capabilities Service **208** that DNS **212** is available in ViBE **202**. MFO **206** may subsequently identify that the resources associated with the ViBE flock config and the DNS flock config are available any may proceed to bootstrap any suitable number of additional resources to the ViBE.

After steps 1-12 are concluded, the process for building the ViBE **202** can be considered complete and the ViBE **202** can be considered built.

FIG. **3** is a block diagram for illustrating an environment **300** and method for bootstrapping services to a target region utilizing the ViBE, according to at least one embodiment.

At step 1, user **302** may utilize any suitable user interface provided by CIOS Central **304** (an example of CIOS Central **108** and CIOS Central **214** of FIGS. **1** and **2**, respectively) to modify region data. By way of example, user **302** may create a new region to which a number of services are to be bootstrapped.

At step 2, CIOS Central **304** may execute operations to send the change to RRDD **306** (e.g., an example of RRDD **104** of FIG. **1**). At step 3, RRDD **306** may store the received region data in database **308**, a data store configured to store region data including any suitable identifier, attribute, state, etc. of a region, AD, realm, ET, or the like. In some embodiments, updater **307** may be utilized to store region data in database **308** or any suitable data store from which such updates may be accessible (e.g., to service teams). In some embodiments, updater **307** may be configured to notify (e.g., via any suitable electronic notification) of updates made to database **308**.

At step 4, MFO **310** (an example of the MFO **106** and **206** of FIGS. **1** and **2**, respectively) may detect the change in region data. In some embodiments, MFO **310** may be configured to poll RRDD **306** for changes in region data. In some embodiments, RRDD **306** may be configured to publish or otherwise notify MFO **310** of region changes.

At step 5, detecting the change in region data may trigger MFO **310** to obtain a version set (e.g., a version set associated with a particular identifier such as a "golden version set" identifier). identifying a particular version for each flock (e.g., service) that is to be bootstrapped to the new region and a particular version for each artifact corresponding to that flock. The version set may be obtained from DB **312**. As flocks evolve and change, the versions for their corresponding configs and artifacts used for region build may change. These changes may be persisted in flock DB **312** such that MFO **310** may identify which versions of flock configs and artifacts to use for building a region (e.g., a ViBE region, a Target Region/non-ViBE Region, etc.). The flock configs (e.g., all versions of the flock configs) and/or artifacts (e.g., all versions of the artifacts) may be stored in DB **308**, DB **312**, or any suitable data store accessible to the CIOS Central **304** and/or MFO **310**.

At step 6, MFO **310** may request CIOS Central **304** to recompile of each of the flock configs associated with the version set with the current region data. In some embodiments, the request may indicate a version for each flock config and/or artifact corresponding to those flock configs.

At step 7, CIOS Central **304** may obtain current region data from the DB **308** (e.g., directly, or via Real-time Regional Data Distributor **306**) and retrieve any suitable flock config and artifact in accordance with the versions requested by MFO **310**.

At step 8, CIOS Central **304** may recompile the flock configs with the region data obtained at step 7 to inject the flock configs with current region data. CIOS Central **304** may return the compiled flock configs to MFO **310**. In some embodiments, CIOS Central **304** may simply indicate compilation is done, and MFO **310** may access the recompiled flock configs via RRDD **306**.

At step 9, MFO **310** may perform a static analysis of the recompiled flock configs. As part of the static analysis, MFO **310** may parse the flock configs (e.g., using a library associated with a declarative infrastructure provisioner (e.g., Terraform, or the like)) to identify dependencies between flocks. From the analysis and the dependencies identified, MFO **310** can generate Build Dependency Graph **338**. Build Dependency Graph **338** may be an acyclic directed graph that identifies an order by which flocks are to be bootstrapped (and/or changes indicated in flock configs are to be applied) to the new region. Each node in the graph may correspond to bootstrapping any suitable portion of a particular flock. The specific bootstrapping order may be identified based at least in part on the dependencies. In some embodiments, the dependencies may be expressed as an attribute of the node and/or indicated via edges of the graph that connect the nodes. MFO **310** may traverse the graph (e.g., beginning at a starting node) to drive the operations of the region build.

In some embodiments, MFO **310** may utilize a cycle detection algorithm to detect the presence of a cycle (e.g., service A depends on service B and vice versa). MFO **310** can identify orphaned capabilities dependencies. For example, MFO **310** can identify orphaned nodes of the Build Dependency Graph **338** that do not connect to any other nodes. MFO **310** may identify falsely published capabilities (e.g., when a capability was prematurely published, and the corresponding functionality is not actually yet available). MFO **310** can detect from the graph that one or more instances of publishing the same capability exist. In some embodiments, any suitable number of these errors may be detected and MFO **310** (or another suitable component such as CIOS Central **304**) may be configured to notify or otherwise present this information to users (e.g., via an electronic notification, a user interface, or the like). In some embodiments, MFO **310** may be configured to force delete/recreate resources to break circular dependencies and may once again provide instructions to CIOS Central **304** to perform bootstrapping operations for those resources and/or corresponding flock configs.

A starting node may correspond to bootstrapping the ViBE flock, a second node may correspond to bootstrapping DNS. The steps 10-15 correspond to deploying (via deployment orchestrator **317**, an example of the deployment orchestrator **218** of FIG. **2**) a ViBE flock to ViBE **316** (e.g., an example of ViBE **116** and **202** of FIGS. **1**, and **2**, respectively). That is, steps 10-15 of FIG. **3** generally correspond to steps 1-6 of FIG. **2**. Once notified that capabilities exist corresponding to the ViBE flock being deployed (e.g., indicating that Capabilities Service **318** and Worker **320**, corresponding to Capabilities Service **208** and Worker **210** of FIG. **2**, are available) the MFO **310** recommence traversal of the Build Dependency Graph **338** to identify next operations to be executed.

By way of example, MFO **310** may continue traversing the Build Dependency Graph **338** to identify that a DNS flock is to be deployed. Steps 16-21 may be executed to deploy DNS **322** (an example of the DNS **212** of FIG. **2**). These operations may generally correspond to steps 7-12 of FIG. **2**.

At step 21, a capability may be stored indicating that DNS **322** is available. Upon detecting this capability, MFO **310** may recommence traversal of the Build Dependency Graph **338**. On this traversal, the MFO **310** may identify that any suitable portion of an instance of CIOS Regional (e.g., an example of CIOS Regional **314**) is to be deployed to the ViBE **316**. In some embodiments, steps 16-21 may be substantially repeated with respect to deploying CIOS Regional (ViBE) **326** (an instance of CIOS Regional **314**, CIOS Regional **110** of FIG. **1**) and Worker **328** to the ViBE **316**. A capability may be transmitted to the Capabilities Service **318** that CIOS Regional (ViBE) **326** is available.

Upon detecting the CIOS Regional (ViBE) **326** is available, MFO **310** may recommence traversal of the Build Dependency Graph **338**. On this traversal, the MFO **310** may identify that a deployment orchestrator (e.g., Deployment Orchestrator **330**, an example of the Deployment Orchestrator **317**) is to be deployed to the ViBE **316**. In some embodiments, steps 16-21 may be substantially repeated with respect to deploying Deployment Orchestrator **330**. Information that identifies a capability may be transmitted to the Capabilities Service **318**, indicating that Deployment Orchestrator **330** is available.

After Deployment Orchestrator **330** is deployed, ViBE **316** may be considered available for processing subsequent requests. Upon detecting Deployment Orchestrator **330** is available, MFO **310** may instruct subsequent bootstrapping requests to be routed to ViBE components rather than utilizing host region components (components of host region **332**). Thus, MFO **310** can continue traversing the Build Dependency Graph **338**, at each node instructing flock deployment to the ViBE **316** via CIOS Central **304**. CIOS Central **304** may request CIOS Regional (ViBE) **326** to deploy resources according to the flock config.

At some point during this process, Target Region **334** may become available. Indication that the Target Region is available may be identifiable from region data for the Target Region **334** being provided by the user **302** (e.g., as an update to the region data). The availability of Target Region **334** may depend on establishing a network connection between the Target Region **334** and external networks (e.g., the Internet). The network connection may be supported over a public network (e.g., the Internet), but use software security tools (e.g., IPsec) to provide one or more encrypted tunnels (e.g., IPsec tunnels such as tunnel **336**) from the ViBE **316** to Target Region **334**. As used herein, "IPsec" refers to a protocol suite for authenticating and encrypting network traffic over a network that uses Internet Protocol (IP) and can include one or more available implementations of the protocol suite (e.g., Openswan, Libreswan, strongSwan, etc.). The network may connect the ViBE **316** to the service enclave of the Target Region **334**.

Prior to establishing the IPsec tunnels, the initial network connection to the Target Region **334** may be on a connection (e.g., an out-of-band VPN tunnel) sufficient to allow bootstrapping of networking services until an IPsec gateway may be deployed on an asset (e.g., bare-metal asset) in the Target Region **334**. To bootstrap the Target Region's **334** network resources, Deployment Orchestrator **330** can deploy the IPsec gateway at the asset within Target Region **334**. The Deployment Orchestrator **330** may then deploy VPN hosts at the Target Region **334** configured to terminate IPsec tunnels from the ViBE **316**. Once services (e.g., Deployment Orchestrator **330**, Service A, etc.) in the ViBE **316** can establish an IPsec connection with the VPN hosts in the Target Region **334**, bootstrapping operations from the ViBE **316** to the Target Region **334** may begin.

In some embodiments, the bootstrapping operations may begin with services in the ViBE **316** provisioning resources in the Target Region **334** to support hosting instances of core services as they are deployed from the ViBE **316**. For example, a host provisioning service may provision hypervisors on infrastructure (e.g., bare-metal hosts) in the Target Region **334** to allocate computing resources for VMs. When the host provisioning service completes allocation of physical resources in the Target Region **334**, the host provisioning service may publish information indicating a capability that indicates that the physical resources in the Target Region **334** have been allocated. The capability may be published to Capabilities Service **318** via CIOS Regional (ViBE) **326** (e.g., by Worker **328**).

With the hardware allocation of the Target Region **334** established and posted to capabilities service **318**, CIOS Regional (ViBE) **326** can orchestrate the deployment of instances of core services from the ViBE **316** to the Target Region **334**. This deployment may be similar to the processes described above for building the ViBE **316**, but using components of the ViBE (e.g., CIOS Regional (ViBE) **326**, Worker **328**, Deployment Orchestrator **330**) instead of components of the Host Region **332** service enclave. The deployment operations may generally correspond to steps 16-21 described above.

As a service is deployed from the ViBE **316** to the Target Region **334**, the DNS record associated with that service may correspond to the instance of the service in the ViBE **316**. The DNS record associated with the service may be updated at a later time to complete deployment of the service to the Target Region **334**. Said another way, the instance of the service in the ViBE **316** may continue to receive traffic (e.g., requests) to the service until the DNS record is updated. A service may deploy partially into the Target Region **334** and publish information indicating a capability (e.g., to Capabilities Service **318**) that the service is partially deployed. For example, a service running in the ViBE **316** may be deployed into the Target Region **334** with a corresponding compute instance, load balancer, and associated applications and other software, but may need to wait for database data to migrate to the Target Region **334** before being completely deployed. The DNS record (e.g., managed by DNS **322**) may still be associated with the service in the ViBE **316**. Once data migration for the service is complete, the DNS record may be updated to point to the operational service deployed in the Target Region **334**. The deployed service in the Target Region **334** may then receive traffic (e.g., requests) for the service, while the instance of the service in the ViBE **316** may no longer receive traffic for the service.

Region Data Management and Usage

FIG. **4** illustrates an example instance of region data **400**, according to at least one embodiment. Region data **400** can be an example of a data object, structure, record, or any suitable type of storage container that includes any suitable number of data definitions corresponding to various portions of region data (e.g., realm attributes, region attributes, execution target attributes, availability domain attributes, etc.). Region data may include any suitable attribute that describes any suitable aspect of the region data. By way of example, region data attributes may include identifiers (e.g., realm identifiers, region identifiers, execution target identifiers, availability domain identifiers, and the like) that may be used to identify and/or differentiate one portion of the region data from another. In some embodiments, region data may include any suitable combination of compliance regulations data (e.g., identifiers indicating applicable compliance regulations such as CSA STAR Certification, etc.), market-place attributes (e.g., marketplace identifiers/names, marketplace type, etc.), geographical-regional attributes (e.g., display names, internal names, etc. identifying aspects of a corresponding geographical region such as "North America," "NORTH AMERICA REGION," and the like), disaster recovery region information (e.g., URLs indicating a storage location of valuable recovery files, information regarding when region data was last updated and by what user, etc.), or the like. Region data may include a region identifier (e.g., an alphanumeric identifier) that may uniquely identify a specific region (e.g., a location corresponding to one or more data centers). Region data **400** may include an identifier (e.g., identifier **401**) with which the attributes of the region data **400** may be accessed and/or referenced.

Region data **400** may include any suitable number of data objects (e.g., structs, lists, arrays, parameters, variables, etc.). As depicted in FIG. **6**, region data **400** may include four data objects (e.g., object **402**, **404**, **406**, and **408**).

Object **402** may be an example of a data object in which one or more sets of realm attributes may be defined. As depicted, object **402** indicates data corresponding to two realms. Realm attribute set **410** is provided as a depiction of a first set of attributes corresponding to a first realm while realm attribute set **412** is provided as a depiction of a second set of attributes corresponding to a second realm that is different from the first realm.

Object **404** may be an example of a data object in which one or more sets of region attributes may be defined. As depicted, object **404** indicates data corresponding to two regions. Region attribute set **414** is provided as a depiction of a first set of attributes corresponding to a first region, while region attribute set **416** is provided as a depiction of a second set of attributes corresponding to a second region that is different from the first region. In some embodiments, the specific attributes included in region attribute set **414** and region attribute set **416** of object **404** may include identifiers corresponding to a respective region.

Object **406** may be another example of a data object in which one or more sets of region attributes may be defined. As depicted, object **406** indicates data corresponding to two regions. Region attribute set **418** is provided as a depiction of a first set of attributes corresponding to a first region while region attribute set **420** is provided as a depiction of a second set of attributes corresponding to a second region that is different from the first region. In some embodiments, region attribute set **418** may include attributes associated with the same region to which region attribute set **414** is associated. The attributes between these two attribute sets may be the same or may differ. As a non-limiting example, region attribute set **418** may include attributes "state" and "ViBE_region" and corresponding values for each. While region attribute set **414** may not include those attributes. Similarly, region attribute set **420** may include attributes "state" and "ViBE_region" and corresponding values for each, while region attribute set **416** may not include those attributes. The specific attributes and/or values provided in any of the objects or attribute sets of the region data **400** may be configurable and/or may vary from those presented in FIG. **4**.

In some embodiments, one or more attributes of an object included in region data **400** may refer to one or more attributes of another object. By way of example, region attribute set **414** of object **404** may indicate an association to realm attribute set **410** based at least in part on inclusion of an attribute "realm" that has a corresponding value "realm **1**" that matches a region attribute set (e.g., realm attribute set **410**) that has the same value associated with its name attribute. In this manner, associations between attribute sets of disparate objects may be maintained.

Object **408** may be another example of a data object in which one or more sets of availability domain (AD) attributes may be defined. As depicted, object **408** indicates data corresponding to two ADs. AD attribute set **422** is provided as a depiction of a first set of attributes corresponding to an AD region while AD attribute set **424** is provided as a depiction of a second set of attributes corresponding to a second AD region that is different from the first AD. In some embodiments, AD attribute set **422** may include attributes associated with the same region to which region attribute set **414** is associated. As depicted AD attribute set **422** may include attributes that are associated with the same realm to which realm attribute set **410** is associated. As depicted, AD attribute set **424** may include attributes associated with the same region to which region attribute set **416** is associated. AD attribute set **424** may include attributes that are associated with the same realm to which realm attribute set **410** is associated. The specific attributes and/or values provided in any of the attribute sets of object **408** need not be the same as the attribute sets provided in object **402-408**.

In some embodiments, each attribute of the object **402-408** may be referenced within one or more configuration files. By way of example, a statement such as "local.realms" may be used to access a list of realms stored in the object **402** and including realm attribute set **410** and **412**.

Any suitable aspect of region data may be defined/specified in a similar manner as depicted in FIG. **4**. These objects may be referenced through various statements and/or parameters in any suitable configuration file of the Cloud Infrastructure Orchestration Service. In some embodiments, region data **400** may be utilized to define/specify any suitable aspect of one or more: realms, regions, ADs, ETs, or the like, any sets of realms, regions, ADs, ETs, etc., and/or relationships/associations between any or all of the above. The region data **400** may be updated in real time (e.g., via the user interface **600** described below in connection with FIG. **6**). In some embodiments, the values of region data **400** may injected into any suitable program code (e.g., one or more configuration files) that references region data **400**. In some embodiments, values of region data **400** may be injected when an object referencing region data **400** is instantiated (e.g., during compilation time). In some embodiments, any suitable compiler injection procedure may be utilized to inject current values of region data **400** into variables referenced in program code.

FIG. **5** illustrates an example of a configuration file (flock config **500**) with one or more parameters referencing region data (e.g., the region data **400** of FIG. **4**), according to at least one embodiment.

A configuration file, or any suitable program code, may reference any suitable portion of predefined region data (e.g., region data **400**). As a non-limiting example, flock config **500** may include a code segment **502** that defines an object (e.g., object **504**) including three variables (e.g., "bootstrap_regions," "production_regions," and "region1_bootstrap_regions") referenced by statements **506**, **508**, and **510**. At compile time (or any suitable time at which the object "locals," corresponding to the region data **400**, are instantiated/created), the variables defined by statements **506-510** may be injected with data obtained from region data **400**. By way of example, the variable "bootstrap_regions" may be injected (e.g., updated) with a list of regions from the list "all_regions" (referencing, by name, object **406** of FIG. **4**) that have a state attribute with a value of "Building," as required in statement **506**. The variable "production_regions" may be injected with a list of regions from the list "all_regions" (referencing, by name, object **406** of FIG. **4**) that have a state attribute with a value of "Production" or "Paused." as required in statement **508**. The variable "region1_bootstrap_regions" may be injected with a list of regions from the list "bootstrap regions" of statement **506** that have a realm attribute with a value of "realm1" or, as required in statement **510**. Thus, compilation of statement **510** may cause the variable "region1_bootstrap_regions" to include a subset of the list of regions injected to variable "bootstrap_regions" (regions for which the state attribute is set to "Building") that also has a realm attributes set to the value "realm **1**."

Once injected with the referenced region data, these variables may be used in any suitable location in the flock config **500** for any suitable purpose. By way of example, statement **512** may instantiate a variable "count" that is set to a length (e.g., an integer) associated with the number of entities provided in the list obtained for the region1_bootstrap_regions variable. As another example, statement **514** may instantiate a variable "name" that may be set to a value corresponding to a name attribute (indicated by ".name") associated with a specific entity (e.g., local.region1_bootstrap_regions[count.index]) within the list obtained for the region1_bootstrap_regions variable.

Any suitable aspect of the region data obtained through instantiation/injection as described herein may be utilized in any suitable manner for any suitable purpose within any suitable configuration file. By utilizing the region data **400** and statements within program code as depicted in FIGS. **4** and **5**, any suitable aspect of region data may be injected into flock configuration files such that manual update of such data is unnecessary. Flock configuration files can instead be written in a region agnostic manner and region data attribute values obtained at compilation time as part of a region build.

FIG. **6** is a block diagram depicting an example user interface **600** for modifying and viewing region data (e.g., region data **400** of FIG. **4**), according to at least one embodiment. Any suitable portion of region data (e.g., any suitable portion of region data **400**) may be presented at user interface (UI) **600**. In some embodiments, the UI **600** may be provided by any suitable component of the Cloud Infrastructure Orchestration Service **102** of FIG. **1**. By way of example, the CIOS Central **108** of FIG. **1** may host the UI **600**.

In some embodiments, UI **600** may present region data **400** in any suitable manner. In some embodiments, UI **600** may be configured to present subsets of the region data in separate portions of the UI **600**. As depicted, UI **600** may present a subset of region data **400** in separate tabs. By way of example, selection of option **602** may cause any suitable data corresponding to "realm1" of the object **402** of FIG. **4** to be presented. Similarly, selection of option **604** may cause any suitable data corresponding to "realm2" of the object **402** to be presented. Any suitable data of any suitable object of region data **400** that is associated with a realm attribute value of "realm1" may be presented within area **606**.

Column **608** may present attribute values corresponding to a particular attribute of objects/attributes sets that are associated with a realm attribute value of "realm **1**." For example, object **404** of FIG. **4** includes a list of attribute sets (e.g., attribute set **414**). In some embodiments, column **608** may be configured to obtain values for a particular object/attribute set (e.g., attribute set **414**) identified with the identifier "public_name." As a specific example, UI **600** may be configured to display "us-seattle-1" at **620** as obtained from the public_name attribute of attribute set **414**.

Column **610** is intended to depict values obtained by "realm1" objects/attributes obtained from region data **400** in a similar manner as described in connection with Column **608**. For purposes of illustration, values displayed via column **610** may present code values (another attribute of the attribute set **414** that was not depicted in FIG. **4** but included in the region data **400**).

Column **612** is intended to depict values obtained by "realm1" objects/attributes obtained from region data **400** that correspond to an attribute "state." For purposes of illustration, values displayed via column **612** may present state value "Production" as provided in attribute set **418**. The value may be obtained based at least in part on identifying the association(s) between attribute sets **418** and **414** (e.g., via matching values for attributes "internal_name" and/or "public name"), and the associations between the attribute sets and a realm with a name attribute set to "realm1."

Any suitable data corresponding to the region data may be presented via UI **600**. In some embodiments, data associated with the region data **400** may be obtained from one or more sources different from the region data **400**. By way of example, column **614** may present ticket information (e.g., a change ticket identifier referring to a specific change made to region data **400**) associated with the region data. In some embodiments, the data presented in column **614** may be obtained from any suitable system such a ticketing system accessible to the Real-time Regional Data Distributor **104** of FIG. **1** (but not depicted in FIG. **1**). In some embodiments, the ticketing system may be a part of CIOS **102** of FIG. **1** or may part of a separate system. The RRDD **104** may obtain data from a ticketing system via function call, application programming interface, or any suitable data transfer method.

As another example, column **618** may correspond to user input provided at any suitable time a corresponding to a particular region. In some embodiments, the user may enter any suitable input (e.g., alphanumeric input) at **622**. RRDD **104** may store this data as part of the region data **400** or as a separate record that is then associated with the region.

In some embodiments, the user may edit any suitable data associated with a given region by selecting an edit option corresponding to that region. By way of example, data for the "us-seattle-1" region may be edited by selecting option **624**. Upon selection, data fields corresponding to the us-seattle-1 region of columns **608-618** may become editable such that changes can be made to the region identifier, code, change ticket, vibe region identifier, and notes, respectively. In some embodiments, selection of the option **624** may make some data fields editable while others remain immutable. By way of example, a data field (e.g., data field **626**) corresponding to us-seattle-1 and column **614** may remain immutable after selection of option **624**. Thus, in some embodiments, at least a portion of the data presented via UI **600** may be unchangeable.

In some embodiments, new region data may be provided via UI **600**. As a non-limiting example, selecting option **628** (or another suitable option, not depicted in FIG. **6**) may enable the user to add/edit data fields within area **606** to correspond to a new region being added to Realm **1**. These data fields may correspond to data fields that may become editable/visible within area **630**. In some embodiments, a separate window or interface (not depicted) may be provided for adding the data fields values corresponding to a new region. In some embodiments, any set of predefined data fields may be provided for editing so as to provide the user to define any suitable attribute of the new region.

FIG. **7** is a block diagram depicting a method for maintaining region state data, according to at least one embodiment. In some embodiments, region state data may indicate one of a set of predefined states (e.g., "none"—state **702**, "Building"—state **704**, "Production"—state **706**, "Paused"—state **708**, and "Deprecated"—state **710**, etc.). In some embodiments, CIOS **102** may transition region state data for a given region between these predefined states according to a predefined protocol and/or in response to user input. The specific states and state transitions depicted in FIG. **7** need not be exhaustive. The particular set of predefined states and particular transitions between the same may differ. As depicted at **712**, state **702** may transition to state **704** according to data indicating an initial stack is up. State **704** may transition to state **702** according to user input indicating the registration of the new region data is canceled as depicted at **713**. State **704** may transition to state **706** according to data indicating a scale-out process is complete as depicted at **714**. State **706** may transition to state **708** according to user input and/or error being received indicative of an operational incident being detected as depicted at **716**. State **708** may transition to state **706** when user input is received indicating the previously detected operational incident is resolved as depicted at **717**. State **708** may transition to state **710** according to user input being received indicating the region is deprecated as depicted at **714**. FIG. **7** describes a method for navigating from one state to another as depicted in FIG. **8**.

FIG. **8** is a block diagram depicting an example method **800** for updating region data, according to at least one embodiment.

At step 1, user **802** may utilize any suitable user interface (e.g., user interface **600** of FIG. **6**, or another suitable user interface) provided by CIOS Central **804** (e.g., an example of CIOS Central **108**, **214**, and/or **304** of FIGS. **1**-**3**, respectively) to modify region data. By way of example, user **802** may create a new region to which a number of services (e.g., "flocks") are to be bootstrapped.

At step 2, CIOS Central **804** may execute operations to send the change to RRDD **806** (e.g., an example of RRDD **104** and **306** of FIGS. **1** and **3**, respectively). At step 3, RRDD **806** may store the received region data in database **808**, a data store configured to store region data including any suitable identifier, attribute, state, etc. of a region, AD, realm, ET, or the like. In some embodiments, updater **807** may be utilized to store region data in database **808** or any suitable data store from which such updates may be accessible (e.g., to service teams). In some embodiments, updater **807** may be configured to notify (e.g., via any suitable electronic notification) of updates made to database **808**.

RRDD **806** may store the received region data in a data store configured to store region data including any suitable identifier, attribute, state, etc. of a region, AD, realm, ET, or the like. By way of example, upon receiving data corresponding to the new region, RRDD **806** may execute operations for creating a new record (e.g., a new object within region data **400** of FIG. **4**). If region data has not existed previously, RRDD **806** may create a new record (e.g., a new object, a table, an object, a container, etc.) within which region data may be maintained. The new record may refer to a new attribute or object corresponding to the new region. As a non-limiting example, adding a new region via UI **600** (using area **630** or a separate window/interface as described above) may cause a new attribute set similar to the region attribute sets **418** and **420** may be added to the object **406** of FIG. **4**. The state attribute for this new attribute set may be set to a value corresponding to state **702** of FIG. **7** (e.g., "none"). This state value may indicate an initial state of the region, prior to any region build being initiated. Any suitable number of objects/attributes may be updated within region data **400** in response to the additional of the new region. The new region data (e.g., the region data **400** as updated) may be stored in DB **808** (an example of the DB **308** of FIG. **3**).

At step 4, MFO **810** (an example of the MFO **106**, **206**, and/or **310** of FIGS. **1**-**3**, respectively) may detect the change in region data. In some embodiments, MFO **810** may be configured to poll (e.g., submit a request to) RRDD **806** for changes in region data. In some embodiments, RRDD **806** may be configured to publish or otherwise notify MFO **810** of region data changes. In some embodiments, RRDD **806** may publish or otherwise notify MFO **810** of region data changes according to any suitable predefined schedule or frequency, upon change, or the like.

At step 5, detecting the change in region data may trigger MFO **810** to obtain a version set (e.g., a version set associated with a particular identifier such as a "golden version set" identifier). identifying a particular version for each flock (e.g., service) that is to be bootstrapped to the new region and a particular version for each artifact corresponding to that flock. The version set may be obtained from DB **812** (an example of the DB **312** of FIG. **3**). As flocks evolve and change, the versions for their corresponding configs and artifacts used for region build may change. These changes may be persisted in flock DB **812** such that MFO **810** may identify which versions of flock configs and artifacts to use for region build. The flock configs (e.g., all versions of the flock configs) and/or artifacts (e.g., all versions of the artifacts) may be stored in DB **808**, DB **812**, or any suitable data store accessible to the CIOS Central **804** and/or MFO **810**.

At step 6, MFO **810** may request CIOS Central **804** to recompile of each of the flock configs associated with the version set with the current region data. In some embodiments, the request may indicate a version for each flock config and/or artifact corresponding to those flock configs.

At step 7, CIOS Central **804** may obtain current region data from the DB **808** (e.g., directly, or via Real-time Regional Data Distributor **806**) and retrieve any suitable flock config and artifact in accordance with the versions requested by MFO **810**.

At step 8, CIOS Central **804** may recompile the flock configs with the region data obtained at step 7 to inject the flock configs with current region data. CIOS Central **804** may return the compiled flock configs to MFO **810**. In some embodiments, CIOS Central **804** may simply indicate compilation is done, and MFO **810** may access the recompiled flock configs via RRDD **806**.

At step 9, MFO **810** may perform a static analysis of the recompiled flock configs. As part of the static analysis, MFO **810** may parse the flock configs (e.g., using a library associated with a component (e.g., a declarative infrastructure provisioner) of CIOS Regional **814**, an example of the CIOS Regional **110** of FIG. **1**) to identify dependencies between resources. From the analysis and the dependencies identified, MFO **810** can generate build dependency graph **814** (an example of the Build dependency graph **338** of FIG. **3**). Build dependency graph **814** may be a directed graph that identifies an order by which flocks are to be bootstrapped to the new region. Each node in the graph may correspond to bootstrapping a particular set of resources. The specific bootstrapping order may be identified based on the dependencies. In some embodiments, the dependencies may be expressed as an attribute of the node and/or indicated via edges of the graph that connect the nodes. MFO **810** may traverse the graph (e.g., beginning at a starting node) to drive the operations of the region build. By way of example, a starting node may correspond to bootstrapping the ViBE flock, a second node may correspond to bootstrapping DNS. The steps 10-15 correspond to deploying (via deployment orchestrator **817**, an example of the deployment orchestrator **317** of FIG. **7**) a ViBE flock to ViBE **816** (e.g., an example of ViBE **116**, **202**, and/or **316** of FIGS. **1**-**3**, respectively). That is, steps 10-15 of FIG. **8** generally correspond to steps 1-6 of FIG. **2**. Once notified that capabilities exist corresponding to the ViBE flock being deployed (e.g., indicating that Capabilities Service **818** and Worker **820**, corresponding to Capabilities Service **318** and Worker **320** of FIG. **3**, are available) the MFO **810** recommences traversal of the build dependency graph **814** to identify subsequent operations to be executed.

By way of example, MFO **810** may traverse the Build Dependency Graph **814** to identify that a DNS flock is to be deployed. Steps 15-20 may be executed to deploy DNS **822** (an example of the DNS **322** of FIG. **3**). These operations may generally correspond to steps 7-12 of FIG. **2**.

At step 14, a capability may be stored indicating that DNS **822** is available. Upon detecting this capability, MFO **810** may recommence traversal of the build dependency graph **814**. On this traversal, the MFO **810** may identify that any suitable portion of an instance of CIOS Regional **110** of FIG. **1** is to be deployed to the ViBE **816**. In some embodiments, steps 15-20 may be substantially repeated with respect to deploying CIOS Regional ViBE **826** (an example of the CIOS Region ViBE **326** of FIG. **3**) and Worker **828** (an example of the Worker **328** of FIG. **3**) to the ViBE **816**. A capability may be transmitted to the Capabilities Service **818** that CIOS Regional (ViBE) **826** and/or worker **828** is available.

At step 21, upon detecting the CIOS Regional **824** and/or worker **828** is available (e.g., via a capability transmitted by or otherwise obtained from Capabilities Service **818**), MFO **810** may update the region data to set the state of the region to state **704** of FIG. **7**, indicating the region is currently being built. In some embodiments, MFO **810** may make a function call (not depicted) to CIOS Central **804** to update the region data (e.g., via RRDD **806**). The updated region data may be persisted in DB **808**. MFO **810** may continue deploying various resources in accordance with its traversal(s) of the build dependency graph **814**. For brevity, these traversals are not depicted in FIG. **8**.

At step 22, MFO **810** may receive/obtain a capability that indicates a last flock has been bootstrapped (e.g., to ViBE **816** or target region **834**). In some embodiments, MFO **810** may automatically update the state associated with the new region to a value indicating state **706** of FIG. **7** (e.g., "Production"). Additionally, or alternatively, user **802** may update the state of the region to the state **706**. In some embodiments, the user **802** may only be provided the option of setting the state value to state **706** after MFO **810** has bootstrapped all services (e.g., within the ViBE **816** or within the target region **834**).

At any suitable time, the user **802** may provide input via the UI **600** (or another suitable interface) that changes the region state to a value indicating state **708** (e.g., "Paused"). State **708** may be used to temporarily pause the region build and/or to prevent functions calls from being made to the new region (e.g., Target Region **834**). In some embodiments, the user **802** may, at any suitable time, modify the region data again to revert the state from state **708** to state **706**.

At any suitable time, the user **802** may provide input via the UI **600** (or another suitable interface) that changes the region state to a value indicating state **710** (e.g., "Deprecated"). In some embodiments, the option to transition the region to state **710** may be provided only on region data that is associated with the state **708** (e.g., "Paused"). State **710** may be used to indicate that the region is no longer supported. While in state **710**, the region will no long appear in any injected variables of the flock configs. In some embodiments, CIOS Central **804** may be configured to skip recompilation of flock configs that explicitly refer to a region in state **710**. In some embodiments, CIOS Central **804** may provide an error or other data for presentation to the user **804** that indicates CIOS Central **804** no longer supports a given region when the region is associated with a state value indicating state **710** ("Deprecated").

FIG. **9** is a block diagram depicting an example method **900** for executing a region build with updated region data, according to at least one embodiment. The method **900** may be performed by one or more components Cloud Infrastructure Orchestration Service **102** of FIG. **1** (e.g., the Real-time Regional Data Distributor **104** of FIG. **1**). A computer-readable storage medium comprising computer-readable instructions that, upon execution by one or more processors of a computing device, cause the computing device to perform the method **900**. The method **900** may performed in any suitable order. It should be appreciated that the method **900** may include a greater number or a lesser number of steps than that depicted in FIG. **9**.

The method may begin at **902**, where region data corresponding to a region of a cloud-computing environment may be maintained (e.g., by Real-time Regional Data Distributor **104** of FIG. **1**). In some embodiments, the region data (e.g., region data **400** of FIG. **6**) may comprise one or more region identifiers and one or more execution target identifiers corresponding to at least one execution target device of the region.

At **904**, a modification of the region data may be detected (e.g., by the RRDD **104** and/or the MFO **106** of FIG. **1**).

At **906**, one or more configuration files corresponding to bootstrapping one or more resources at corresponding execution target devices of the region may be obtained (e.g., by MFO **106**, by CIOS Central **108** of FIG. **1**, etc.). In some embodiments, the one or more configuration files utilize parameters that reference respective portions of the region data.

At **908**, operations to cause the one or more configuration files to be updated may be executed by the Cloud Infrastructure Orchestration Service based at least in part on detecting the modification of the region data. For example, the CIOS Central **108** may recompile the one or more configuration files, causing the updated region data to be injected to those configuration files.

At **910**, a region build associated with bootstrapping one or more resources at corresponding execution target devices of the region may be executed based at least in part on the one or more configuration files as updated.

Example Cloud Service Infrastructure Architecture

As noted above, infrastructure as a service (IaaS) is one particular type of cloud computing. IaaS can be configured to provide virtualized computing resources over a public network (e.g., the Internet). In an IaaS model, a cloud computing provider can host the infrastructure components (e.g., servers, storage devices, network nodes (e.g., hardware), deployment software, platform virtualization (e.g., a hypervisor layer), or the like). In some cases, an IaaS provider may also supply a variety of services to accompany those infrastructure components (e.g., billing, monitoring, logging, load balancing and clustering, etc.). Thus, as these services may be policy-driven, IaaS users may be able to implement policies to drive load balancing to maintain application availability and performance.

In some instances, IaaS customers may access resources and services through a wide area network (WAN), such as the Internet, and can use the cloud provider's services to install the remaining elements of an application stack. For example, the user can log in to the IaaS platform to create virtual machines (VMs), install operating systems (OSs) on each VM, deploy middleware such as databases, create storage buckets for workloads and backups, and even install enterprise software into that VM. Customers can then use the provider's services to perform various functions, including balancing network traffic, troubleshooting application issues, monitoring performance, managing disaster recovery, etc.

In most cases, a cloud computing model will require the participation of a cloud provider. The cloud provider may, but need not be, a third-party service that specializes in providing (e.g., offering, renting, selling) IaaS. An entity might also opt to deploy a private cloud, becoming its own provider of infrastructure services.

In some examples, IaaS deployment is the process of putting a new application, or a new version of an application, onto a prepared application server or the like. It may also include the process of preparing the server (e.g., installing libraries, daemons, etc.). This is often managed by the cloud provider, below the hypervisor layer (e.g., the servers, storage, network hardware, and virtualization). Thus, the customer may be responsible for handling (OS), middleware, and/or application deployment (e.g., on self-service virtual machines (e.g., that can be spun up on demand) or the like.

In some examples, IaaS provisioning may refer to acquiring computers or virtual hosts for use, and even installing needed libraries or services on them. In most cases, deployment does not include provisioning, and the provisioning may need to be performed first.

In some cases, there are two different challenges for IaaS provisioning. First, there is the initial challenge of provisioning the initial set of infrastructure before anything is running. Second, there is the challenge of evolving the existing infrastructure (e.g., adding new services, changing services, removing services, etc.) once everything has been provisioned. In some cases, these two challenges may be addressed by enabling the configuration of the infrastructure to be defined declaratively. In other words, the infrastructure (e.g., what components are needed and how they interact) can be defined by one or more configuration files. Thus, the overall topology of the infrastructure (e.g., what resources depend on which, and how they each work together) can be described declaratively. In some instances, once the topology is defined, a workflow can be generated that creates and/or manages the different components described in the configuration files.

In some examples, an infrastructure may have many interconnected elements. For example, there may be one or more virtual private clouds (VPCs) (e.g., a potentially on-demand pool of configurable and/or shared computing resources), also known as a core network. In some examples, there may also be one or more inbound/outbound traffic group rules provisioned to define how the inbound and/or outbound traffic of the network will be set up and one or more virtual machines (VMs). Other infrastructure elements may also be provisioned, such as a load balancer, a database, or the like. As more and more infrastructure elements are desired and/or added, the infrastructure may incrementally evolve.

In some instances, continuous deployment techniques may be employed to enable deployment of infrastructure code across various virtual computing environments. Additionally, the described techniques can enable infrastructure management within these environments. In some examples, service teams can write code that is desired to be deployed to one or more, but often many, different production environments (e.g., across various different geographic locations, sometimes spanning the entire world). However, in some examples, the infrastructure on which the code will be deployed must first be set up. In some instances, the provisioning can be done manually, a provisioning tool may be utilized to provision the resources, and/or deployment tools may be utilized to deploy the code once the infrastructure is provisioned.

FIG. **10** is a block diagram **1000** illustrating an example pattern of an IaaS architecture, according to at least one embodiment. Service operators **1002** can be communicatively coupled to a secure host tenancy **1004** that can include a virtual cloud network (VCN) **1006** and a secure host subnet **1008**. In some examples, the service operators **1002** may be using one or more client computing devices, which may be portable handheld devices (e.g., an iPhone®, cellular telephone, an iPad®, computing tablet, a personal digital assistant (PDA)) or wearable devices (e.g., a Google Glass® head mounted display), running software such as Microsoft Windows Mobile®, and/or a variety of mobile operating systems such as iOS, Windows Phone, Android, BlackBerry 8, Palm OS, and the like, and being Internet, e-mail, short message service (SMS), Blackberry®, or other communication protocol enabled. Alternatively, the client computing devices can be general purpose personal computers including, by way of example, personal computers and/or laptop computers running various versions of Microsoft Windows®, Apple Macintosh®, and/or Linux operating systems. The client computing devices can be workstation computers running any of a variety of commercially-available UNIX® or UNIX-like operating systems, including without limitation the variety of GNU/Linux operating systems, such as for example, Google Chrome OS. Alternatively, or in addition, client computing devices may be any other electronic device, such as a thin-client computer, an Internet-enabled gaming system (e.g., a Microsoft Xbox gaming console with or without a Kinect® gesture input device), and/or a personal messaging device, capable of communicating over a network that can access the VCN **1006** and/or the Internet.

The VCN **1006** can include a local peering gateway (LPG) **1010** that can be communicatively coupled to a secure shell (SSH) VCN **1012** via an LPG **1010** contained in the SSH VCN **1012**. The SSH VCN **1012** can include an SSH subnet **1014**, and the SSH VCN **1012** can be communicatively coupled to a control plane VCN **1016** via the LPG **1010** contained in the control plane VCN **1016**. Also, the SSH VCN **1012** can be communicatively coupled to a data plane VCN **1018** via an LPG **1010**. The control plane VCN **1016** and the data plane VCN **1018** can be contained in a service tenancy **1019** that can be owned and/or operated by the IaaS provider.

The control plane VCN **1016** can include a control plane demilitarized zone (DMZ) tier **1020** that acts as a perimeter network (e.g., portions of a corporate network between the corporate intranet and external networks). The DMZ-based servers may have restricted responsibilities and help keep breaches contained. Additionally, the DMZ tier **1020** can include one or more load balancer (LB) subnet(s) **1022**, a control plane app tier **1024** that can include app subnet(s) **1026**, a control plane data tier **1028** that can include database (DB) subnet(s) **1030** (e.g., frontend DB subnet(s) and/or backend DB subnet(s)). The LB subnet(s) **1022** contained in the control plane DMZ tier **1020** can be communicatively coupled to the app subnet(s) **1026** contained in the control plane app tier **1024** and an Internet gateway **1034** that can be contained in the control plane VCN **1016**, and the app subnet(s) **1026** can be communicatively coupled to the DB subnet(s) **1030** contained in the control plane data tier **1028** and a service gateway **1036** and a network address translation (NAT) gateway **1038**. The control plane VCN **1016** can include the service gateway **1036** and the NAT gateway **1038**.

The control plane VCN **1016** can include a data plane mirror app tier **1040** that can include app subnet(s) **1026**. The app subnet(s) **1026** contained in the data plane mirror app tier **1040** can include a virtual network interface controller (VNIC) **1042** that can execute a compute instance **1044**. The compute instance **1044** can communicatively couple the app subnet(s) **1026** of the data plane mirror app tier **1040** to app subnet(s) **1026** that can be contained in a data plane app tier **1046**.

The data plane VCN **1018** can include the data plane app tier **1046**, a data plane DMZ tier **1048**, and a data plane data tier **1050**. The data plane DMZ tier **1048** can include LB subnet(s) **1022** that can be communicatively coupled to the app subnet(s) **1026** of the data plane app tier **1046** and the Internet gateway **1034** of the data plane VCN **1018**. The app subnet(s) **1026** can be communicatively coupled to the service gateway **1036** of the data plane VCN **1018** and the NAT gateway **1038** of the data plane VCN **1018**. The data plane data tier **1050** can also include the DB subnet(s) **1030** that can be communicatively coupled to the app subnet(s) **1026** of the data plane app tier **1046**.

The Internet gateway **1034** of the control plane VCN **1016** and of the data plane VCN **1018** can be communicatively coupled to a metadata management service **1052** that can be communicatively coupled to public Internet **1054**. Public Internet **1054** can be communicatively coupled to the NAT gateway **1038** of the control plane VCN **1016** and of the data plane VCN **1018**. The service gateway **1036** of the control plane VCN **1016** and of the data plane VCN **1018** can be communicatively couple to cloud services **1056**.

In some examples, the service gateway **1036** of the control plane VCN **1016** or of the data plane VCN **1018** can make application programming interface (API) calls to cloud services **1056** without going through public Internet **1054**. The API calls to cloud services **1056** from the service gateway **1036** can be one-way: the service gateway **1036** can make API calls to cloud services **1056**, and cloud services **1056** can send requested data to the service gateway **1036**. But, cloud services **1056** may not initiate API calls to the service gateway **1036**.

In some examples, the secure host tenancy **1004** can be directly connected to the service tenancy **1019**, which may be otherwise isolated. The secure host subnet **1008** can communicate with the SSH subnet **1014** through an LPG **1010** that may enable two-way communication over an otherwise isolated system. Connecting the secure host subnet **1008** to the SSH subnet **1014** may give the secure host subnet **1008** access to other entities within the service tenancy **1019**.

The control plane VCN **1016** may allow users of the service tenancy **1019** to set up or otherwise provision desired resources. Desired resources provisioned in the control plane VCN **1016** may be deployed or otherwise used in the data plane VCN **1018**. In some examples, the control plane VCN **1016** can be isolated from the data plane VCN **1018**, and the data plane mirror app tier **1040** of the control plane VCN **1016** can communicate with the data plane app tier **1046** of the data plane VCN **1018** via VNICs **1042** that can be contained in the data plane mirror app tier **1040** and the data plane app tier **1046**.

In some examples, users of the system, or customers, can make requests, for example create, read, update, or delete (CRUD) operations, through public Internet **1054** that can communicate the requests to the metadata management service **1052**. The metadata management service **1052** can communicate the request to the control plane VCN **1016** through the Internet gateway **1034**. The request can be received by the LB subnet(s) **1022** contained in the control plane DMZ tier **1020**. The LB subnet(s) **1022** may determine that the request is valid, and in response to this determination, the LB subnet(s) **1022** can transmit the request to app subnet(s) **1026** contained in the control plane app tier **1024**. If the request is validated and requires a call to public Internet **1054**, the call to public Internet **1054** may be transmitted to the NAT gateway **1038** that can make the call to public Internet **1054**. Memory that may be desired to be stored by the request can be stored in the DB subnet(s) **1030**.

In some examples, the data plane mirror app tier **1040** can facilitate direct communication between the control plane VCN **1016** and the data plane VCN **1018**. For example, changes, updates, or other suitable modifications to configuration may be desired to be applied to the resources contained in the data plane VCN **1018**. Via a VNIC **1042**, the control plane VCN **1016** can directly communicate with, and can thereby execute the changes, updates, or other suitable modifications to configuration to, resources contained in the data plane VCN **1018**.

In some embodiments, the control plane VCN **1016** and the data plane VCN **1018** can be contained in the service tenancy **1019**. In this case, the user, or the customer, of the system may not own or operate either the control plane VCN **1016** or the data plane VCN **1018**. Instead, the IaaS provider may own or operate the control plane VCN **1016** and the data plane VCN **1018**, both of which may be contained in the service tenancy **1019**. This embodiment can enable isolation of networks that may prevent users or customers from interacting with other users', or other customers', resources. Also, this embodiment may allow users or customers of the system to store databases privately without needing to rely on public Internet **1054**, which may not have a desired level of threat prevention, for storage.

In other embodiments, the LB subnet(s) **1022** contained in the control plane VCN **1016** can be configured to receive a signal from the service gateway **1036**. In this embodiment, the control plane VCN **1016** and the data plane VCN **1018** may be configured to be called by a customer of the IaaS provider without calling public Internet **1054**. Customers of the IaaS provider may desire this embodiment since database(s) that the customers use may be controlled by the IaaS provider and may be stored on the service tenancy **1019**, which may be isolated from public Internet **1054**.

FIG. **11** is a block diagram **1100** illustrating another example pattern of an IaaS architecture, according to at least one embodiment. Service operators **1102** (e.g., service operators **1002** of FIG. **10**) can be communicatively coupled to a secure host tenancy **1104** (e.g., the secure host tenancy **1004** of FIG. **10**) that can include a virtual cloud network (VCN) **1106** (e.g., the VCN **1006** of FIG. **10**) and a secure host subnet **1108** (e.g., the secure host subnet **1008** of FIG. **10**). The VCN **1106** can include a local peering gateway (LPG) **1110** (e.g., the LPG **1010** of FIG. **10**) that can be communicatively coupled to a secure shell (SSH) VCN **1112** (e.g., the SSH VCN **1012** of FIG. **10**) via an LPG **1010** contained in the SSH VCN **1112**. The SSH VCN **1112** can include an SSH subnet **1114** (e.g., the SSH subnet **1014** of FIG. **10**), and the SSH VCN **1112** can be communicatively coupled to a control plane VCN **1116** (e.g., the control plane VCN **1016** of FIG. **10**) via an LPG **1110** contained in the control plane VCN **1116**. The control plane VCN **1116** can be contained in a service tenancy **1119** (e.g., the service tenancy **1019** of FIG. **10**), and the data plane VCN **1118** (e.g., the data plane VCN **1018** of FIG. **10**) can be contained in a customer tenancy **1121** that may be owned or operated by users, or customers, of the system.

The control plane VCN **1116** can include a control plane DMZ tier **1120** (e.g., the control plane DMZ tier **1020** of FIG. **10**) that can include LB subnet(s) **1122** (e.g., LB subnet(s) **1022** of FIG. **10**), a control plane app tier **1124** (e.g., the control plane app tier **1024** of FIG. **10**) that can include app subnet(s) **1126** (e.g., app subnet(s) **1026** of FIG. **10**), a control plane data tier **1128** (e.g., the control plane data tier **1028** of FIG. **10**) that can include database (DB) subnet(s) **1130** (e.g., similar to DB subnet(s) **1030** of FIG. **10**). The LB subnet(s) **1122** contained in the control plane DMZ tier **1120** can be communicatively coupled to the app subnet(s) **1126** contained in the control plane app tier **1124** and an Internet gateway **1134** (e.g., the Internet gateway **1034** of FIG. **10**) that can be contained in the control plane VCN **1116**, and the app subnet(s) **1126** can be communicatively coupled to the DB subnet(s) **1130** contained in the control plane data tier **1128** and a service gateway **1136** (e.g., the service gateway **1036** of FIG. **10**) and a network address translation (NAT) gateway **1138** (e.g., the NAT gateway **1038** of FIG. **10**). The control plane VCN **1116** can include the service gateway **1136** and the NAT gateway **1138**.

The control plane VCN **1116** can include a data plane mirror app tier **1140** (e.g., the data plane mirror app tier **1040** of FIG. **10**) that can include app subnet(s) **1126**. The app subnet(s) **1126** contained in the data plane mirror app tier **1140** can include a virtual network interface controller (VNIC) **1142** (e.g., the VNIC of **1042**) that can execute a compute instance **1144** (e.g., similar to the compute instance **1044** of FIG. **10**). The compute instance **1144** can facilitate communication between the app subnet(s) **1126** of the data plane mirror app tier **1140** and the app subnet(s) **1126** that can be contained in a data plane app tier **1146** (e.g., the data plane app tier **1046** of FIG. **10**) via the VNIC **1142** contained in the data plane mirror app tier **1140** and the VNIC **1142** contained in the data plane app tier **1146**.

The Internet gateway **1134** contained in the control plane VCN **1116** can be communicatively coupled to a metadata management service **1152** (e.g., the metadata management service **1052** of FIG. **10**) that can be communicatively coupled to public Internet **1154** (e.g., public Internet **1054** of FIG. **10**). Public Internet **1154** can be communicatively coupled to the NAT gateway **1138** contained in the control plane VCN **1116**. The service gateway **1136** contained in the control plane VCN **1116** can be communicatively couple to cloud services **1156** (e.g., cloud services **1056** of FIG. **10**).

In some examples, the data plane VCN **1118** can be contained in the customer tenancy **1121**. In this case, the IaaS provider may provide the control plane VCN **1116** for each customer, and the IaaS provider may, for each customer, set up a unique compute instance **1144** that is contained in the service tenancy **1119**. Each compute instance **1144** may allow communication between the control plane VCN **1116**, contained in the service tenancy **1119**, and the data plane VCN **1118** that is contained in the customer tenancy **1121**. The compute instance **1144** may allow resources, that are provisioned in the control plane VCN **1116** that is contained in the service tenancy **1119**, to be deployed or otherwise used in the data plane VCN **1118** that is contained in the customer tenancy **1121**.

In other examples, the customer of the IaaS provider may have databases that live in the customer tenancy **1121**. In this example, the control plane VCN **1116** can include the data plane mirror app tier **1140** that can include app subnet(s) **1126**. The data plane mirror app tier **1140** can reside in the data plane VCN **1118**, but the data plane mirror app tier **1140** may not live in the data plane VCN **1118**. That is, the data plane mirror app tier **1140** may have access to the customer tenancy **1121**, but the data plane mirror app tier **1140** may not exist in the data plane VCN **1118** or be owned or operated by the customer of the IaaS provider. The data plane mirror app tier **1140** may be configured to make calls to the data plane VCN **1118** but may not be configured to make calls to any entity contained in the control plane VCN **1116**. The customer may desire to deploy or otherwise use resources in the data plane VCN **1118** that are provisioned in the control plane VCN **1116**, and the data plane mirror app tier **1140** can facilitate the desired deployment, or other usage of resources, of the customer.

In some embodiments, the customer of the IaaS provider can apply filters to the data plane VCN **1118**. In this embodiment, the customer can determine what the data plane VCN **1118** can access, and the customer may restrict access to public Internet **1154** from the data plane VCN **1118**. The IaaS provider may not be able to apply filters or otherwise control access of the data plane VCN **1118** to any outside networks or databases. Applying filters and controls by the customer onto the data plane VCN **1118**, contained in the customer tenancy **1121**, can help isolate the data plane VCN **1118** from other customers and from public Internet **1154**.

In some embodiments, cloud services **1156** can be called by the service gateway **1136** to access services that may not exist on public Internet **1154**, on the control plane VCN **1116**, or on the data plane VCN **1118**. The connection between cloud services **1156** and the control plane VCN **1116** or the data plane VCN **1118** may not be live or continuous. Cloud services **1156** may exist on a different network owned or operated by the IaaS provider. Cloud services **1156** may be configured to receive calls from the service gateway **1136** and may be configured to not receive calls from public Internet **1154**. Some cloud services **1156** may be isolated from other cloud services **1156**, and the control plane VCN **1116** may be isolated from cloud services **1156** that may not be in the same region as the control plane VCN **1116**. For example, the control plane VCN **1116** may be located in "Region 1," and cloud service "Deployment 10," may be located in Region 1 and in "Region 2." If a call to Deployment 10 is made by the service gateway **1136** contained in the control plane VCN **1116** located in Region 1, the call may be transmitted to Deployment 10 in Region 1. In this example, the control plane VCN **1116**, or Deployment 10 in Region 1, may not be communicatively coupled to, or otherwise in communication with, Deployment 10 in Region 2.

FIG. **12** is a block diagram **1200** illustrating another example pattern of an IaaS architecture, according to at least one embodiment. Service operators **1202** (e.g., service operators **1002** of FIG. **10**) can be communicatively coupled to a secure host tenancy **1204** (e.g., the secure host tenancy **1004** of FIG. **10**) that can include a virtual cloud network (VCN) **1206** (e.g., the VCN **1006** of FIG. **10**) and a secure host subnet **1208** (e.g., the secure host subnet **1008** of FIG. **10**). The VCN **1206** can include an LPG **1210** (e.g., the LPG **1010** of FIG. **10**) that can be communicatively coupled to an SSH VCN **1212** (e.g., the SSH VCN **1012** of FIG. **10**) via an LPG **1210** contained in the SSH VCN **1212**. The SSH VCN **1212** can include an SSH subnet **1214** (e.g., the SSH subnet **1014** of FIG. **10**), and the SSH VCN **1212** can be communicatively coupled to a control plane VCN **1216** (e.g., the control plane VCN **1016** of FIG. **10**) via an LPG **1210** contained in the control plane VCN **1216** and to a data plane VCN **1218** (e.g., the data plane **1018** of FIG. **10**) via an LPG **1210** contained in the data plane VCN **1218**. The control plane VCN **1216** and the data plane VCN **1218** can be contained in a service tenancy **1219** (e.g., the service tenancy **1019** of FIG. **10**).

The control plane VCN **1216** can include a control plane DMZ tier **1220** (e.g., the control plane DMZ tier **1020** of FIG. **10**) that can include load balancer (LB) subnet(s) **1222** (e.g., LB subnet(s) **1022** of FIG. **10**), a control plane app tier **1224** (e.g., the control plane app tier **1024** of FIG. **10**) that can include app subnet(s) **1226** (e.g., similar to app subnet(s) **1026** of FIG. **10**), a control plane data tier **1228** (e.g., the control plane data tier **1028** of FIG. **10**) that can include DB subnet(s) **1230**. The LB subnet(s) **1222** contained in the control plane DMZ tier **1220** can be communicatively coupled to the app subnet(s) **1226** contained in the control plane app tier **1224** and to an Internet gateway **1234** (e.g., the Internet gateway **1034** of FIG. **10**) that can be contained in the control plane VCN **1216**, and the app subnet(s) **1226** can be communicatively coupled to the DB subnet(s) **1230** contained in the control plane data tier **1228** and to a service gateway **1236** (e.g., the service gateway of FIG. **10**) and a network address translation (NAT) gateway **1238** (e.g., the NAT gateway **1038** of FIG. **10**). The control plane VCN **1216** can include the service gateway **1236** and the NAT gateway **1238**.

The data plane VCN **1218** can include a data plane app tier **1246** (e.g., the data plane app tier **1046** of FIG. **10**), a data plane DMZ tier **1248** (e.g., the data plane DMZ tier **1048** of FIG. **10**), and a data plane data tier **1250** (e.g., the data plane data tier **1050** of FIG. **10**). The data plane DMZ tier **1248** can include LB subnet(s) **1222** that can be communicatively coupled to trusted app subnet(s) **1260** and untrusted app subnet(s) **1262** of the data plane app tier **1246** and the Internet gateway **1234** contained in the data plane VCN **1218**. The trusted app subnet(s) **1260** can be communicatively coupled to the service gateway **1236** contained in the data plane VCN **1218**, the NAT gateway **1238** contained in the data plane VCN **1218**, and DB subnet(s) **1230** contained in the data plane data tier **1250**. The untrusted app subnet(s) **1262** can be communicatively coupled to the service gateway **1236** contained in the data plane VCN **1218** and DB subnet(s) **1230** contained in the data plane data tier **1250**. The data plane data tier **1250** can include DB subnet(s) **1230** that can be communicatively coupled to the service gateway **1236** contained in the data plane VCN **1218**.

The untrusted app subnet(s) **1262** can include one or more primary VNICs **1264**(**1**)-(N) that can be communicatively coupled to tenant virtual machines (VMs) **1266**(**1**)-(N). Each tenant VM **1266**(**1**)-(N) can be communicatively coupled to a respective app subnet **1267**(**1**)-(N) that can be contained in respective container egress VCNs **1268**(**1**)-(N) that can be contained in respective customer tenancies **1270**(**1**)-(N). Respective secondary VNICs **1272**(**1**)-(N) can facilitate communication between the untrusted app subnet(s) **1262** contained in the data plane VCN **1218** and the app subnet contained in the container egress VCNs **1268**(**1**)-(N). Each container egress VCNs **1268**(**1**)-(N) can include a NAT gateway **1238** that can be communicatively coupled to public Internet **1254** (e.g., public Internet **1054** of FIG. **10**).

The Internet gateway **1234** contained in the control plane VCN **1216** and contained in the data plane VCN **1218** can be communicatively coupled to a metadata management service **1252** (e.g., the metadata management system **1052** of FIG. **10**) that can be communicatively coupled to public Internet **1254**. Public Internet **1254** can be communicatively coupled to the NAT gateway **1238** contained in the control plane VCN **1216** and contained in the data plane VCN **1218**. The service gateway **1236** contained in the control plane VCN **1216** and contained in the data plane VCN **1218** can be communicatively couple to cloud services **1256**.

In some embodiments, the data plane VCN **1218** can be integrated with customer tenancies **1270**. This integration can be useful or desirable for customers of the IaaS provider in some cases such as a case that may desire support when executing code. The customer may provide code to run that may be destructive, may communicate with other customer resources, or may otherwise cause undesirable effects. In response to this, the IaaS provider may determine whether to run code given to the IaaS provider by the customer.

In some examples, the customer of the IaaS provider may grant temporary network access to the IaaS provider and request a function to be attached to the data plane app tier **1246**. Code to run the function may be executed in the VMs **1266**(**1**)-(N), and the code may not be configured to run anywhere else on the data plane VCN **1218**. Each VM **1266**(**1**)-(N) may be connected to one customer tenancy **1270**. Respective containers **1271**(**1**)-(N) contained in the VMs **1266**(**1**)-(N) may be configured to run the code. In this case, there can be a dual isolation (e.g., the containers **1271**(**1**)-(N) running code, where the containers **1271**(**1**)-(N) may be contained in at least the VM **1266**(**1**)-(N) that are contained in the untrusted app subnet(s) **1262**), which may help prevent incorrect or otherwise undesirable code from damaging the network of the IaaS provider or from damaging a network of a different customer. The containers **1271** (**1**)-(N) may be communicatively coupled to the customer tenancy **1270** and may be configured to transmit or receive data from the customer tenancy **1270**. The containers **1271** (**1**)-(N) may not be configured to transmit or receive data from any other entity in the data plane VCN **1218**. Upon completion of running the code, the IaaS provider may kill or otherwise dispose of the containers **1271**(**1**)-(N).

In some embodiments, the trusted app subnet(s) **1260** may run code that may be owned or operated by the IaaS provider. In this embodiment, the trusted app subnet(s) **1260** may be communicatively coupled to the DB subnet(s) **1230** and be configured to execute CRUD operations in the DB subnet(s) **1230**. The untrusted app subnet(s) **1262** may be communicatively coupled to the DB subnet(s) **1230**, but in this embodiment, the untrusted app subnet(s) may be configured to execute read operations in the DB subnet(s) **1230**. The containers **1271**(**1**)-(N) that can be contained in the VM **1266**(**1**)-(N) of each customer and that may run code from the customer may not be communicatively coupled with the DB subnet(s) **1230**.

In other embodiments, the control plane VCN **1216** and the data plane VCN **1218** may not be directly communicatively coupled. In this embodiment, there may be no direct communication between the control plane VCN **1216** and the data plane VCN **1218**. However, communication can occur indirectly through at least one method. An LPG **1210** may be established by the IaaS provider that can facilitate communication between the control plane VCN **1216** and the data plane VCN **1218**. In another example, the control plane VCN **1216** or the data plane VCN **1218** can make a call to cloud services **1256** via the service gateway **1236**. For example, a call to cloud services **1256** from the control plane VCN **1216** can include a request for a service that can communicate with the data plane VCN **1218**.

FIG. **13** is a block diagram **1300** illustrating another example pattern of an IaaS architecture, according to at least one embodiment. Service operators **1302** (e.g., service operators **1002** of FIG. **10**) can be communicatively coupled to a secure host tenancy **1304** (e.g., the secure host tenancy **1004** of FIG. **10**) that can include a virtual cloud network (VCN) **1306** (e.g., the VCN **1006** of FIG. **10**) and a secure host subnet **1308** (e.g., the secure host subnet **1008** of FIG. **10**). The VCN **1306** can include an LPG **1310** (e.g., the LPG **1010** of FIG. **10**) that can be communicatively coupled to an SSH VCN **1312** (e.g., the SSH VCN **1012** of FIG. **10**) via an LPG **1310** contained in the SSH VCN **1312**. The SSH VCN **1312** can include an SSH subnet **1314** (e.g., the SSH subnet **1014** of FIG. **10**), and the SSH VCN **1312** can be communicatively coupled to a control plane VCN **1316** (e.g., the control plane VCN **1016** of FIG. **10**) via an LPG **1310** contained in the control plane VCN **1316** and to a data plane VCN **1318** (e.g., the data plane **1018** of FIG. **10**) via an LPG **1310** contained in the data plane VCN **1318**. The control plane VCN **1316** and the data plane VCN **1318** can be contained in a service tenancy **1319** (e.g., the service tenancy **1019** of FIG. **10**).

The control plane VCN **1316** can include a control plane DMZ tier **1320** (e.g., the control plane DMZ tier **1020** of FIG. **10**) that can include LB subnet(s) **1322** (e.g., LB subnet(s) **1022** of FIG. **10**), a control plane app tier **1324** (e.g., the control plane app tier **1024** of FIG. **10**) that can include app subnet(s) **1326** (e.g., app subnet(s) **1026** of FIG. **10**), a control plane data tier **1328** (e.g., the control plane data tier **1028** of FIG. **10**) that can include DB subnet(s) **1330** (e.g., DB subnet(s) **1230** of FIG. **12**). The LB subnet(s) **1322** contained in the control plane DMZ tier **1320** can be communicatively coupled to the app subnet(s) **1326** contained in the control plane app tier **1324** and to an Internet gateway **1334** (e.g., the Internet gateway **1034** of FIG. **10**) that can be contained in the control plane VCN **1316**, and the app subnet(s) **1326** can be communicatively coupled to the DB subnet(s) **1330** contained in the control plane data tier **1328** and to a service gateway **1336** (e.g., the service gateway of FIG. **10**) and a network address translation (NAT) gateway **1338** (e.g., the NAT gateway **1038** of FIG. **10**). The control plane VCN **1316** can include the service gateway **1336** and the NAT gateway **1338**.

The data plane VCN **1318** can include a data plane app tier **1346** (e.g., the data plane app tier **1046** of FIG. **10**), a data plane DMZ tier **1348** (e.g., the data plane DMZ tier **1048** of FIG. **10**), and a data plane data tier **1350** (e.g., the data plane data tier **1050** of FIG. **10**). The data plane DMZ tier **1348** can include LB subnet(s) **1322** that can be communicatively coupled to trusted app subnet(s) **1360** (e.g., trusted app subnet(s) **1260** of FIG. **12**) and untrusted app subnet(s) **1362** (e.g., untrusted app subnet(s) **1262** of FIG. **12**) of the data plane app tier **1346** and the Internet gateway **1334** contained in the data plane VCN **1318**. The trusted app subnet(s) **1360** can be communicatively coupled to the service gateway **1336** contained in the data plane VCN **1318**, the NAT gateway **1338** contained in the data plane VCN **1318**, and DB subnet(s) **1330** contained in the data plane data tier **1350**. The untrusted app subnet(s) **1362** can be communicatively coupled to the service gateway **1336** contained in the data plane VCN **1318** and DB subnet(s) **1330** contained in the data plane data tier **1350**. The data plane data tier **1350** can include DB subnet(s) **1330** that can be communicatively coupled to the service gateway **1336** contained in the data plane VCN **1318**.

The untrusted app subnet(s) **1362** can include primary VNICs **1364(1)**-(N) that can be communicatively coupled to tenant virtual machines (VMs) **1366(1)**-(N) residing within the untrusted app subnet(s) **1362**. Each tenant VM **1366(1)**-(N) can run code in a respective container **1367(1)**-(N) and be communicatively coupled to an app subnet **1326** that can be contained in a data plane app tier **1346** that can be contained in a container egress VCN **1368**. Respective secondary VNICs **1372(1)**-(N) can facilitate communication between the untrusted app subnet(s) **1362** contained in the data plane VCN **1318** and the app subnet contained in the container egress VCN **1368**. The container egress VCN can include a NAT gateway **1338** that can be communicatively coupled to public Internet **1354** (e.g., public Internet **1054** of FIG. **10**).

The Internet gateway **1334** contained in the control plane VCN **1316** and contained in the data plane VCN **1318** can be communicatively coupled to a metadata management service **1352** (e.g., the metadata management system **1052** of FIG. **10**) that can be communicatively coupled to public Internet **1354**. Public Internet **1354** can be communicatively coupled to the NAT gateway **1338** contained in the control plane VCN **1316** and contained in the data plane VCN **1318**. The service gateway **1336** contained in the control plane VCN **1316** and contained in the data plane VCN **1318** can be communicatively couple to cloud services **1356**.

In some examples, the pattern illustrated by the architecture of block diagram **1300** of FIG. **13** may be considered an exception to the pattern illustrated by the architecture of block diagram **1200** of FIG. **12** and may be desirable for a customer of the IaaS provider if the IaaS provider cannot directly communicate with the customer (e.g., a disconnected region). The respective containers **1367(1)**-(N) that are contained in the VMs **1366(1)**-(N) for each customer can be accessed in real-time by the customer. The containers **1367(1)**-(N) may be configured to make calls to respective secondary VNICs **1372(1)**-(N) contained in app subnet(s) **1326** of the data plane app tier **1346** that can be contained in the container egress VCN **1368**. The secondary VNICs **1372(1)**-(N) can transmit the calls to the NAT gateway **1338** that may transmit the calls to public Internet **1354**. In this example, the containers **1367(1)**-(N) that can be accessed in real-time by the customer can be isolated from the control plane VCN **1316** and can be isolated from other entities contained in the data plane VCN **1318**. The containers **1367(1)**-(N) may also be isolated from resources from other customers.

In other examples, the customer can use the containers **1367(1)**-(N) to call cloud services **1356**. In this example, the customer may run code in the containers **1367(1)**-(N) that requests a service from cloud services **1356**. The containers **1367(1)**-(N) can transmit this request to the secondary VNICs **1372(1)**-(N) that can transmit the request to the NAT gateway that can transmit the request to public Internet **1354**. Public Internet **1354** can transmit the request to LB subnet(s) **1322** contained in the control plane VCN **1316** via the Internet gateway **1334**. In response to determining the request is valid, the LB subnet(s) can transmit the request to app subnet(s) **1326** that can transmit the request to cloud services **1356** via the service gateway **1336**.

It should be appreciated that IaaS architectures **1000**, **1100**, **1200**, **1300** depicted in the figures may have other components than those depicted. Further, the embodiments shown in the figures are only some examples of a cloud infrastructure system that may incorporate an embodiment of the disclosure. In some other embodiments, the IaaS systems may have more or fewer components than shown in the figures, may combine two or more components, or may have a different configuration or arrangement of components.

In certain embodiments, the IaaS systems described herein may include a suite of applications, middleware, and database service offerings that are delivered to a customer in a self-service, subscription-based, elastically scalable, reliable, highly available, and secure manner. An example of such an IaaS system is the Oracle Cloud Infrastructure (OCI) provided by the present assignee.

FIG. **14** illustrates an example computer system **1400**, in which various embodiments may be implemented. The system **1400** may be used to implement any of the computer systems described above. As shown in the figure, computer system **1400** includes a processing unit **1404** that communicates with a number of peripheral subsystems via a bus subsystem **1402**. These peripheral subsystems may include a processing acceleration unit **1406**, an I/O subsystem **1408**, a storage subsystem **1418** and a communications subsystem **1424**. Storage subsystem **1418** includes tangible computer-readable storage media **1422** and a system memory **1410**.

Bus subsystem **1402** provides a mechanism for letting the various components and subsystems of computer system **1400** communicate with each other as intended. Although bus subsystem **1402** is shown schematically as a single bus, alternative embodiments of the bus subsystem may utilize multiple buses. Bus subsystem **1402** may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. For example, such architectures may include an Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus, which can be implemented as a Mezzanine bus manufactured to the IEEE P1386.1 standard.

Processing unit **1404**, which can be implemented as one or more integrated circuits (e.g., a conventional microprocessor or microcontroller), controls the operation of computer system **1400**. One or more processors may be included in processing unit **1404**. These processors may include single core or multicore processors. In certain embodiments, processing unit **1404** may be implemented as one or more independent processing units **1432** and/or **1434** with single or multicore processors included in each processing unit. In other embodiments, processing unit **1404** may also be implemented as a quad-core processing unit formed by integrating two dual-core processors into a single chip.

In various embodiments, processing unit **1404** can execute a variety of programs in response to program code and can maintain multiple concurrently executing programs or processes. At any given time, some or all of the program code to be executed can be resident in processor(s) **1404** and/or in storage subsystem **1418**. Through suitable programming, processor(s) **1404** can provide various functionalities described above. Computer system **1400** may additionally include a processing acceleration unit **1406**, which can include a digital signal processor (DSP), a special-purpose processor, and/or the like.

I/O subsystem **1408** may include user interface input devices and user interface output devices. User interface input devices may include a keyboard, pointing devices such as a mouse or trackball, a touchpad or touch screen incorporated into a display, a scroll wheel, a click wheel, a dial, a button, a switch, a keypad, audio input devices with voice command recognition systems, microphones, and other types of input devices. User interface input devices may include, for example, motion sensing and/or gesture recognition devices such as the Microsoft Kinect® motion sensor that enables users to control and interact with an input device, such as the Microsoft Xbox® 360 game controller, through a natural user interface using gestures and spoken commands. User interface input devices may also include eye gesture recognition devices such as the Google Glass® blink detector that detects eye activity (e.g., 'blinking' while taking pictures and/or making a menu selection) from users and transforms the eye gestures as input into an input device (e.g., Google Glass®). Additionally, user interface input devices may include voice recognition sensing devices that enable users to interact with voice recognition systems (e.g., Siri® navigator), through voice commands.

User interface input devices may also include, without limitation, three dimensional (3D) mice, joysticks or pointing sticks, gamepads and graphic tablets, and audio/visual devices such as speakers, digital cameras, digital camcorders, portable media players, webcams, image scanners, fingerprint scanners, barcode reader 3D scanners, 3D printers, laser rangefinders, and eye gaze tracking devices. Additionally, user interface input devices may include, for example, medical imaging input devices such as computed tomography, magnetic resonance imaging, position emission tomography, medical ultrasonography devices. User interface input devices may also include, for example, audio input devices such as MIDI keyboards, digital musical instruments and the like.

User interface output devices may include a display subsystem, indicator lights, or non-visual displays such as audio output devices, etc. The display subsystem may be a cathode ray tube (CRT), a flat-panel device, such as that using a liquid crystal display (LCD) or plasma display, a projection device, a touch screen, and the like. In general, use of the term "output device" is intended to include all possible types of devices and mechanisms for outputting information from computer system **1400** to a user or other computer. For example, user interface output devices may include, without limitation, a variety of display devices that visually convey text, graphics and audio/video information such as monitors, printers, speakers, headphones, automotive navigation systems, plotters, voice output devices, and modems.

Computer system **1400** may comprise a storage subsystem **1418** that provides a tangible non-transitory computer-readable storage medium for storing software and data constructs that provide the functionality of the embodiments described in this disclosure. The software can include programs, code modules, instructions, scripts, etc., that when executed by one or more cores or processors of processing unit **1404** provide the functionality described above. Storage subsystem **1418** may also provide a repository for storing data used in accordance with the present disclosure.

As depicted in the example in FIG. **14**, storage subsystem **1418** can include various components including a system memory **1410**, computer-readable storage media **1422**, and a computer readable storage media reader **1420**. System memory **1410** may store program instructions that are loadable and executable by processing unit **1404**. System memory **1410** may also store data that is used during the execution of the instructions and/or data that is generated during the execution of the program instructions. Various different kinds of programs may be loaded into system memory **1410** including but not limited to client applications, Web browsers, mid-tier applications, relational database management systems (RDBMS), virtual machines, containers, etc.

System memory **1410** may also store an operating system **1416**. Examples of operating system **1416** may include various versions of Microsoft Windows®, Apple Macintosh®, and/or Linux operating systems, a variety of commercially-available UNIX® or UNIX-like operating systems (including without limitation the variety of GNU/Linux operating systems, the Google Chrome® OS, and the like) and/or mobile operating systems such as iOS, Windows® Phone, Android® OS, BlackBerry® OS, and Palm® OS operating systems. In certain implementations where computer system **1400** executes one or more virtual machines, the virtual machines along with their guest operating systems (GOSs) may be loaded into system memory **1410** and executed by one or more processors or cores of processing unit **1404**.

System memory **1410** can come in different configurations depending upon the type of computer system **1400**. For example, system memory **1410** may be volatile memory (such as random access memory (RAM)) and/or non-volatile memory (such as read-only memory (ROM), flash memory, etc.) Different types of RAM configurations may be provided including a static random access memory (SRAM), a dynamic random access memory (DRAM), and others. In some implementations, system memory **1410** may include a basic input/output system (BIOS) containing basic routines that help to transfer information between elements within computer system **1400**, such as during start-up.

Computer-readable storage media **1422** may represent remote, local, fixed, and/or removable storage devices plus storage media for temporarily and/or more permanently containing, storing, computer-readable information for use by computer system **1400** including instructions executable by processing unit **1404** of computer system **1400**.

Computer-readable storage media **1422** can include any appropriate media known or used in the art, including storage media and communication media, such as but not limited to, volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage and/or transmission of information. This can include tangible computer-readable storage media such as RAM, ROM, electronically erasable programmable ROM (EEPROM), flash memory or other memory technology, CD-ROM, digital versatile disk (DVD), or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or other tangible computer readable media.

By way of example, computer-readable storage media **1422** may include a hard disk drive that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive that reads from or writes to a removable, non-volatile magnetic disk, and an optical disk drive that reads from or writes to a removable, nonvolatile optical disk such as a CD ROM, DVD, and Blu-Ray® disk, or other optical media. Computer-readable storage media **1422** may include, but is not limited to, Zip® drives, flash memory cards, universal serial bus (USB) flash drives, secure digital (SD) cards, DVD disks, digital video tape, and the like. Computer-readable storage media **1422** may also include, solid-state drives (SSD) based on non-volatile memory such as flash-memory based SSDs, enterprise flash drives, solid state ROM, and the like, SSDs based on volatile memory such as solid state RAM, dynamic RAM, static RAM, DRAM-based SSDs, magnetoresistive RAM (MRAM) SSDs, and hybrid SSDs that use a combination of DRAM and flash memory based SSDs. The disk drives and their associated computer-readable media may provide non-volatile storage of computer-readable instructions, data structures, program modules, and other data for computer system **1400**.

Machine-readable instructions executable by one or more processors or cores of processing unit **1404** may be stored on a non-transitory computer-readable storage medium. A non-transitory computer-readable storage medium can include physically tangible memory or storage devices that include volatile memory storage devices and/or non-volatile storage devices. Examples of non-transitory computer-readable storage medium include magnetic storage media (e.g., disk or tapes), optical storage media (e.g., DVDs, CDs), various types of RAM, ROM, or flash memory, hard drives, floppy drives, detachable memory drives (e.g., USB drives), or other type of storage device.

Communications subsystem **1424** provides an interface to other computer systems and networks. Communications subsystem **1424** serves as an interface for receiving data from and transmitting data to other systems from computer system **1400**. For example, communications subsystem **1424** may enable computer system **1400** to connect to one or more devices via the Internet. In some embodiments communications subsystem **1424** can include radio frequency (RF) transceiver components for accessing wireless voice and/or data networks (e.g., using cellular telephone technology, advanced data network technology, such as 3G, 4G or EDGE (enhanced data rates for global evolution), WiFi (IEEE 802.11 family standards, or other mobile communication technologies, or any combination thereof), global positioning system (GPS) receiver components, and/or other components. In some embodiments communications subsystem **1424** can provide wired network connectivity (e.g., Ethernet) in addition to or instead of a wireless interface.

In some embodiments, communications subsystem **1424** may also receive input communication in the form of structured and/or unstructured data feeds **1426**, event streams **1428**, event updates **1430**, and the like on behalf of one or more users who may use computer system **1400**.

By way of example, communications subsystem **1424** may be configured to receive data feeds **1426** in real-time from users of social networks and/or other communication services such as Twitter® feeds, Facebook® updates, web feeds such as Rich Site Summary (RSS) feeds, and/or real-time updates from one or more third party information sources.

Additionally, communications subsystem **1424** may also be configured to receive data in the form of continuous data streams, which may include event streams **1428** of real-time events and/or event updates **1430**, that may be continuous or unbounded in nature with no explicit end. Examples of applications that generate continuous data may include, for example, sensor data applications, financial tickers, network performance measuring tools (e.g., network monitoring and traffic management applications), clickstream analysis tools, automobile traffic monitoring, and the like.

Communications subsystem **1424** may also be configured to output the structured and/or unstructured data feeds **1426**, event streams **1428**, event updates **1430**, and the like to one or more databases that may be in communication with one or more streaming data source computers coupled to computer system **1400**.

Computer system **1400** can be one of various types, including a handheld portable device (e.g., an iPhone® cellular phone, an iPad® computing tablet, a PDA), a wearable device (e.g., a Google Glass® head mounted display), a PC, a workstation, a mainframe, a kiosk, a server rack, or any other data processing system.

Due to the ever-changing nature of computers and networks, the description of computer system **1400** depicted in the figure is intended only as a specific example. Many other configurations having more or fewer components than the system depicted in the figure are possible. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, firmware, software (including applets), or a combination. Further, connection to other computing devices, such as network input/output devices, may be employed. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the various embodiments.

Although specific embodiments have been described, various modifications, alterations, alternative constructions, and equivalents are also encompassed within the scope of the disclosure. Embodiments are not restricted to operation within certain specific data processing environments but are free to operate within a plurality of data processing environments. Additionally, although embodiments have been described using a particular series of transactions and steps, it should be apparent to those skilled in the art that the scope of the present disclosure is not limited to the described series of transactions and steps. Various features and aspects of the above-described embodiments may be used individually or jointly.

Further, while embodiments have been described using a particular combination of hardware and software, it should be recognized that other combinations of hardware and software are also within the scope of the present disclosure. Embodiments may be implemented only in hardware, or only in software, or using combinations thereof. The various processes described herein can be implemented on the same processor or different processors in any combination. Accordingly, where components or modules are described as being configured to perform certain operations, such configuration can be accomplished, e.g., by designing electronic circuits to perform the operation, by programming programmable electronic circuits (such as microprocessors) to perform the operation, or any combination thereof. Processes can communicate using a variety of techniques including but not limited to conventional techniques for inter process communication, and different pairs of processes may use different techniques, or the same pair of processes may use different techniques at different times.

The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that additions, subtractions, deletions, and other modifications and changes may be made thereunto without departing from the broader spirit and scope as set forth in the claims. Thus, although specific disclosure embodiments have been described, these are not intended to be limiting. Various modifications and equivalents are within the scope of the following claims.

The use of the terms "a" and "an" and "the" and similar referents in the context of describing the disclosed embodiments (especially in the context of the following claims) are to be construed to cover both the singular and the plural, unless otherwise indicated herein or clearly contradicted by context. The terms "comprising," "having," "including," and "containing" are to be construed as open-ended terms (i.e., meaning "including, but not limited to,") unless otherwise noted. The term "connected" is to be construed as partly or wholly contained within, attached to, or joined together, even if there is something intervening. Recitation of ranges of values herein are merely intended to serve as a shorthand method of referring individually to each separate value falling within the range, unless otherwise indicated herein and each separate value is incorporated into the specification as if it were individually recited herein. All methods described herein can be performed in any suitable order unless otherwise indicated herein or otherwise clearly contradicted by context. The use of any and all examples, or exemplary language (e.g., "such as") provided herein, is intended merely to better illuminate embodiments and does not pose a limitation on the scope of the disclosure unless otherwise claimed. No language in the specification should be construed as indicating any non-claimed element as essential to the practice of the disclosure.

Disjunctive language such as the phrase "at least one of X, Y, or Z," unless specifically stated otherwise, is intended to be understood within the context as used in general to present that an item, term, etc., may be either X, Y, or Z, or any combination thereof (e.g., X, Y, and/or Z). Thus, such disjunctive language is not generally intended to, and should not, imply that certain embodiments require at least one of X, at least one of Y, or at least one of Z to each be present.

Preferred embodiments of this disclosure are described herein, including the best mode known for carrying out the disclosure. Variations of those preferred embodiments may become apparent to those of ordinary skill in the art upon reading the foregoing description. Those of ordinary skill should be able to employ such variations as appropriate and the disclosure may be practiced otherwise than as specifically described herein. Accordingly, this disclosure includes all modifications and equivalents of the subject matter recited in the claims appended hereto as permitted by applicable law. Moreover, any combination of the above-described elements in all possible variations thereof is encompassed by the disclosure unless otherwise indicated herein.

All references, including publications, patent applications, and patents, cited herein are hereby incorporated by reference to the same extent as if each reference were individually and specifically indicated to be incorporated by reference and were set forth in its entirety herein.

In the foregoing specification, aspects of the disclosure are described with reference to specific embodiments thereof, but those skilled in the art will recognize that the disclosure is not limited thereto. Various features and aspects of the above-described disclosure may be used individually or jointly. Further, embodiments can be utilized in any number of environments and applications beyond those described herein without departing from the broader spirit and scope of the specification. The specification and drawings are, accordingly, to be regarded as illustrative rather than restrictive.

What is claimed is:

1. A computer-implemented method, comprising:

maintaining, by a cloud infrastructure orchestration service, region data corresponding to a region of a cloud-computing environment, the region data being maintained in a data structure that specifies one or more region identifiers and one or more execution target identifiers corresponding to at least one execution target device of the region;

detecting, by the cloud infrastructure orchestration service, a modification of the region data;

obtaining, by the cloud infrastructure orchestration service, one or more configuration files corresponding to bootstrapping one or more resources at corresponding execution target devices of the region, a configuration file of the one or more configuration files comprising a statement that references the data structure;

generating, by the cloud infrastructure orchestration service, a build dependency graph that specifies an order for executing tasks associated with bootstrapping the one or more resources, the build dependency graph being generated based on parsing the statement that references the data structure, wherein parsing the statement causes the configuration file to be updated with the region data, and wherein the region data is utilized to determine the order and tasks that the build dependency graph specifies; and

executing, by the cloud infrastructure orchestration service, a region build associated with bootstrapping the one or more resources according to the region data, the region build being executed in accordance with the order and tasks specified by the build dependency graph.

2. The computer-implemented method of claim **1**, wherein detecting the modification to the region data further comprises:

providing one or more user interface for modifying the region data;

receiving, at the one or more user interfaces, user input comprising the modification to the region data; and

updating the region data in accordance with the user input.

3. The computer-implemented method of claim **1**, further comprising maintaining, by a Real-time Regional Data

Distributor of the cloud infrastructure orchestration service, the region data in a persisted record, the region data further identifying at least one of: an availability domain, an instance of region data corresponding to a virtual bootstrap environment of the cloud infrastructure orchestration service, or a realm identifier.

4. The computer-implemented method of claim **1**, further comprising:

maintaining, by the cloud infrastructure orchestration service, a state associated with the region build; and

presenting the state of the region build at a user interface provided by the cloud infrastructure orchestration service.

5. The computer-implemented method of claim **1**, comprising performing operations to cause the one or more configuration files to be updated based at least in part on recompiling the one or more configuration files, thereby injecting the one or more configuration files with updated region data.

6. The computer-implemented method of claim **1**, wherein bootstrapping the one or more resources within the region comprises provisioning at least one infrastructure component and deploying at least one artifact to the at least one infrastructure component in accordance with one or more configuration files comprising the region data as updated.

7. A computing system of a cloud-computing environment, comprising:

one or more processors; and

one or more non-transitory computer-readable storage media comprising computer-readable instructions that, when executed by the one or more processors, cause the computing system to:

maintain region data corresponding to a region of the cloud-computing environment, the region data being maintained in a data structure that specifies one or more region identifiers and one or more execution target identifiers corresponding to at least one execution target device of the region;

detect a modification of the region data;

obtain one or more configuration files corresponding to bootstrapping one or more resources at corresponding execution target devices of the region, a configuration file of the one or more configuration files comprising a statement that references the data structure;

execute operations to generate a build dependency graph that specifies an order for executing tasks associated with bootstrapping the one or more resources, the build dependency graph being generated based on parsing the statement that references the data structure, wherein parsing the statement causes the configuration file to be updated with the region data, and wherein the region data is utilized to determine the order and tasks that the build dependency graph specifies; and

execute a region build associated with bootstrapping one or more services with the region according to the region data, the region build being executed in accordance with the order and tasks specified by the build dependency graph.

8. The computing system of claim **7**, wherein executing corresponding operations to detect the modification to the region data further causes the computing system to:

provide one or more user interface for modifying the region data;

receive, at the one or more user interfaces, user input comprising the modification to the region data; and

update the region data in accordance with the user input.

9. The computing system of claim **7**, wherein executing the computer-readable instructions further causes the computing system to maintain the region data in a persisted record, the region data further identifying at least one of: an availability domain, an instance of region data corresponding to a virtual bootstrap environment of the cloud-computing environment, or a realm identifier.

10. The computing system of claim **7**, wherein executing the computer-readable instructions further causes the computing system to:

maintain a state associated with the region build; and

present the state of the region build at a user interface.

11. The computing system of claim **7**, wherein executing the computer-readable instructions further causes the computer system perform operations causing the one or more configuration files to be updated based at least in part on recompiling the one or more configuration files, thereby injecting the one or more configuration files with the region data.

12. The computing system of claim **7**, wherein executing the computer-readable instructions to bootstrap the one or more services within the region further causes the computing system to provision at least one infrastructure component and deploy at least one artifact to the at least one infrastructure component in accordance with one or more configuration files comprising the region data as updated.

13. A non-transitory computer-readable storage medium storing computer-readable instructions that, when executed by one or more processors of a cloud-computing environment, cause a cloud infrastructure orchestration service of the cloud-computing environment to:

maintain region data corresponding to a region of the cloud-computing environment, the region data being maintained in a data structure that specifies one or more region identifiers and one or more execution target identifiers corresponding to at least one execution target device of the region;

detect a modification of the region data;

obtain one or more configuration files corresponding to bootstrapping one or more resources at corresponding execution target devices of the region, a configuration file of the one or more configuration files comprising a statement that references the data structure;

execute operations to generate a build dependency graph that specifies an order for executing tasks associated with bootstrapping the one or more resources, the build dependency graph being generated based on parsing the statement that references the data structure, wherein parsing the statement causes the configuration file to be updated with the region data, and wherein the region data is utilized to determine the order and tasks that the build dependency graph specifies; and

execute a region build associated with bootstrapping one or more services with the region according to the region data, the region build being executed in accordance with the order and tasks specified by the build dependency graph.

14. The non-transitory computer-readable storage medium of claim **13**, wherein executing the operations to detect the modification to the region data further causes the cloud infrastructure orchestration service to:

provide one or more user interface for modifying the region data;

receive, at the one or more user interfaces, user input comprising the modification to the region data; and

update the region data in accordance with the user input.

15. The non-transitory computer-readable storage medium of claim **13**, wherein executing the computer-readable instructions further causes the cloud infrastructure orchestration service to maintain the region data in a persisted record, the region data further identifying at least one of: an availability domain, an instance of region data corresponding to a virtual bootstrap environment of the cloud infrastructure orchestration service, or a realm identifier.

16. The non-transitory computer-readable storage medium of claim **13**, wherein executing the computer-readable instructions further causes the cloud infrastructure orchestration service to:

maintain a state associated with the region build; and

present the state of the region build at a user interface provided by the cloud infrastructure orchestration service.

17. The non-transitory computer-readable storage medium of claim **13**, wherein executing the computer-readable instructions further causes the one or more processors to perform operations causing the one or more configuration files to be updated based at least in part on recompiling the one or more configuration files, thereby injecting the one or more configuration files with the region data.

18. The non-transitory computer-readable storage medium of claim **13**, wherein executing the computer-readable instructions to bootstrap the one or more resources within the region further causes the cloud infrastructure orchestration service to provision at least one infrastructure component and deploy at least one artifact to the at least one infrastructure component in accordance with one or more configuration files comprising the region data as updated.

19. The computer-implemented method of claim **1**, wherein parsing the statement of the configuration file generates an instance of the data structure, and wherein generating the instance of the data structure causes the region data to be utilized to determine the order and tasks specified by the build dependency graph.

20. The computer-implemented method of claim **1**, wherein executing the region build comprises traversing the build dependency graph.

* * * * *