DevOps: Reliability, Monitoring and Management with Service Asset and Configuration Management

Yuri Ardulov, Principal System Architect
RingCentral Inc., San Mateo, CA, USA
yuri@ardmail.com

Dmitry Shchemelinin, Philosophy Doctor, Sr. Director of Operations
RingCentral Inc., San Mateo, CA, USA
dmitry.shchemelinin@ringcentral.com

Abstract

Many organizations focus on how to ensure the reliability, manageability and security of both their critical infrastructure and their core application base. To stay competitive, these organizations need to minimize the risk of service interruptions when they upgrade systems, introduce new features or retire the old one. These interruptions can affect the business and the reputation of the firm, especially when this is a 24x7x365 service. By reaching a certain size of the customer base as well as infrastructure, old approaches stop working. **In this article we present our journey into the DevOps Service Management world and SACM (Service Asset and Configuration Management) system and processes, which we are building to monitor and control our distributed environment to deliver quality services that are aligned with the business goals and objectives**.

1. Table of contents

1. RC architecture

1.1. About RC

RingCentral (RC) is an international IT company with a central office in Silicon Valley (Belmont, CA, USA), provisioning voice over Internet protocol (VoIP), mobile platforms, short message service (SMS), email, fax, conference and many other IP communication services for more than 500,000 business customers in the USA, Canada, and Europe [1].
The RC production system is a cloud based multi-component infrastructure with big data traffic across all the distributed environments. Figure 1 shows the RC scalable architecture located in four data centers on the West Coast and East Coast of the USA and in Western Europe. The RC environments, both production and stress test, are growing rapidly along with customer's demand, having up to a 40% annual rate, and now consist of more than 10,000 hosts, including hardware (HW) but mostly virtual machines (VM) for more efficient IT maintenance and saving computing resources [2].
All production servers are grouped into approximately 60 pools and RC components, each provisioning a particular custom service and functionality or connecting with external public switched telephone network (PSTN) and other 3rd party providers.

1.2. Current architectural concepts

Current service architecture is presented below (Fig. 1.2.1), and it reflects the distributed nature of RC services.

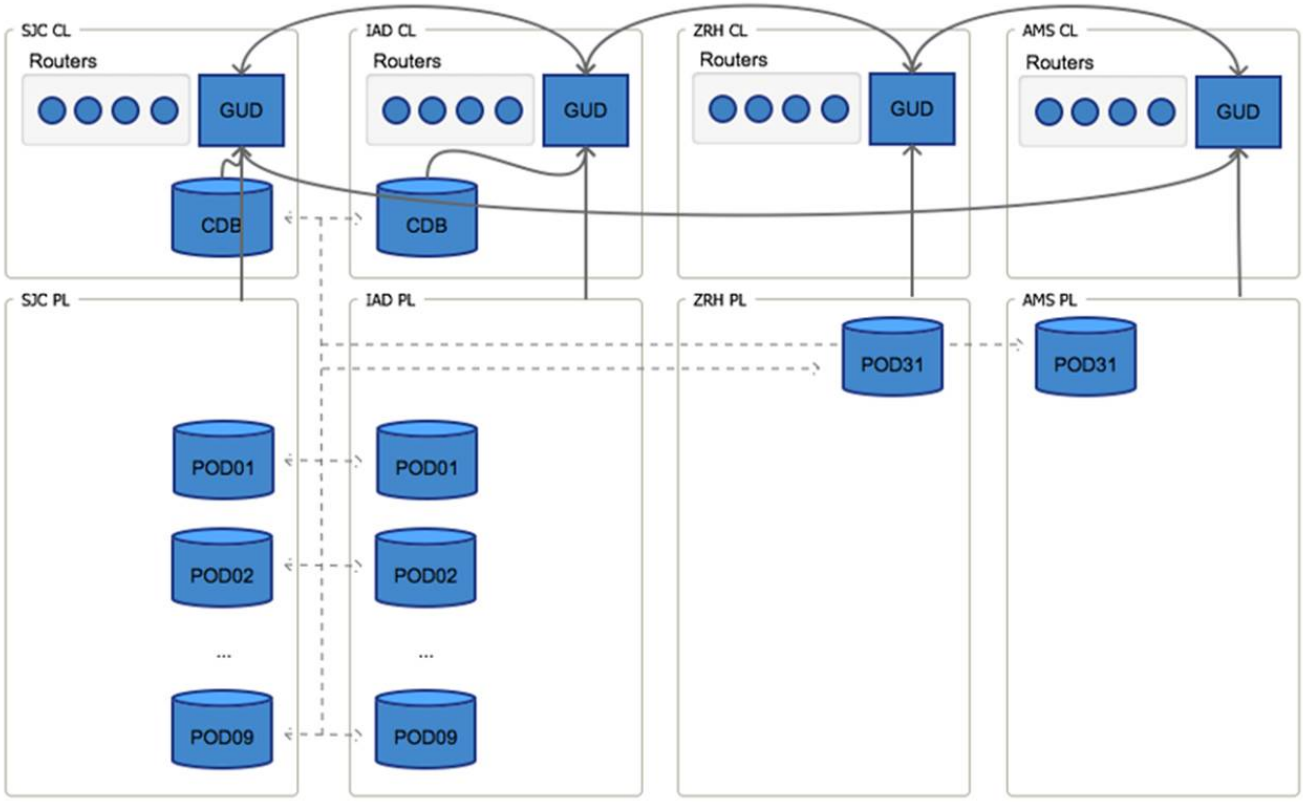


Fig. 1.2.1

Based on the figure above we have two logical groupings,:

- Common Layer (CL) – is the layer which provides full communication between all geo-locations
- POD Layer (PL) – is the layer that acts in the active/passive mode between geo-locations for resilience purpose. The service relationship across geo-deployment is presented on the (Fig. 1.2.2)

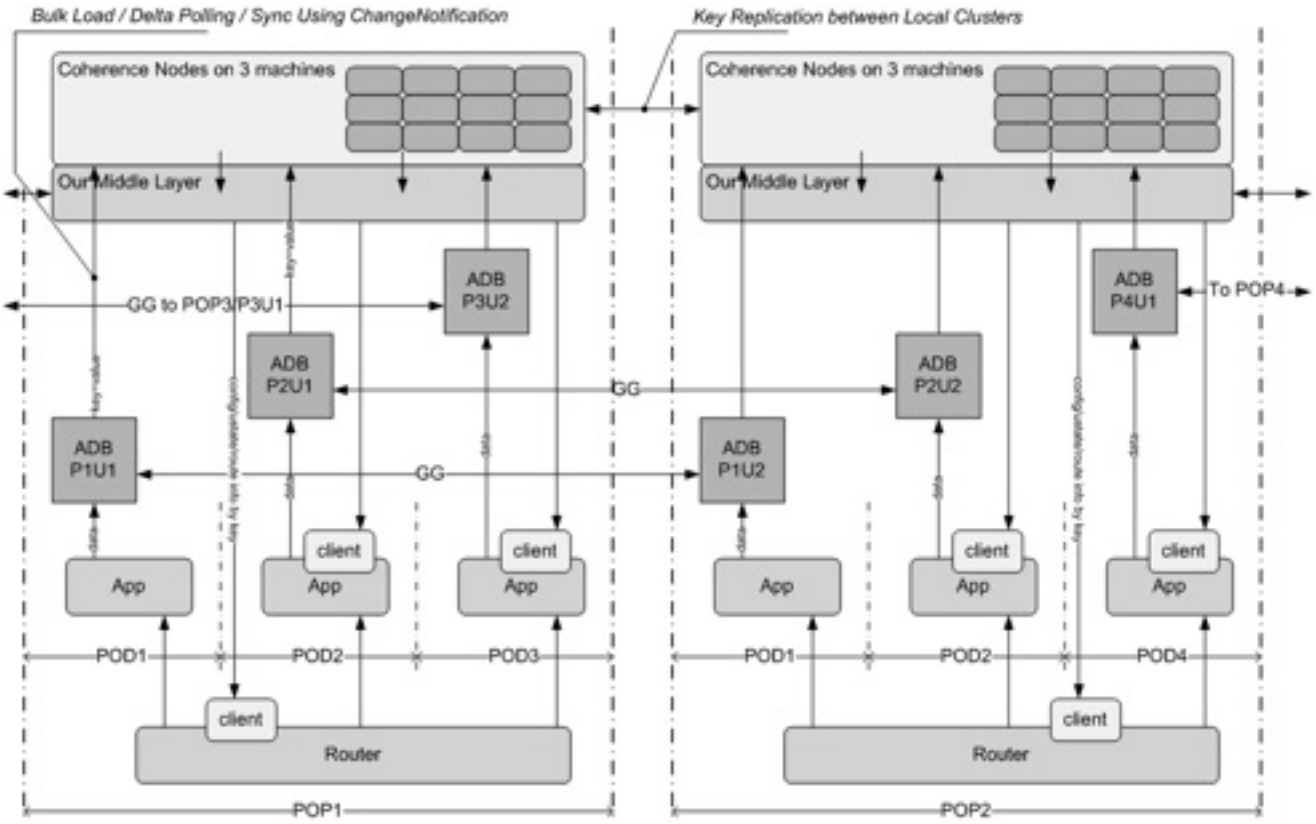


Fig 1.2.2.

Persistent information is replicated between active and passive locations and it provides the pivotal element of the service, and is presented in Fig 1.2.3. This presents Tier1 of the service, which comprises the prime business of the company. Additionally, there are Tier[2-4] types of services, which provide different mechanisms and processes around the core service business. For example Tier2 services are presented in Fig. 1.2.4. These services are distributed and have a level of redundancy, which allows operations to continuously manage all core services.

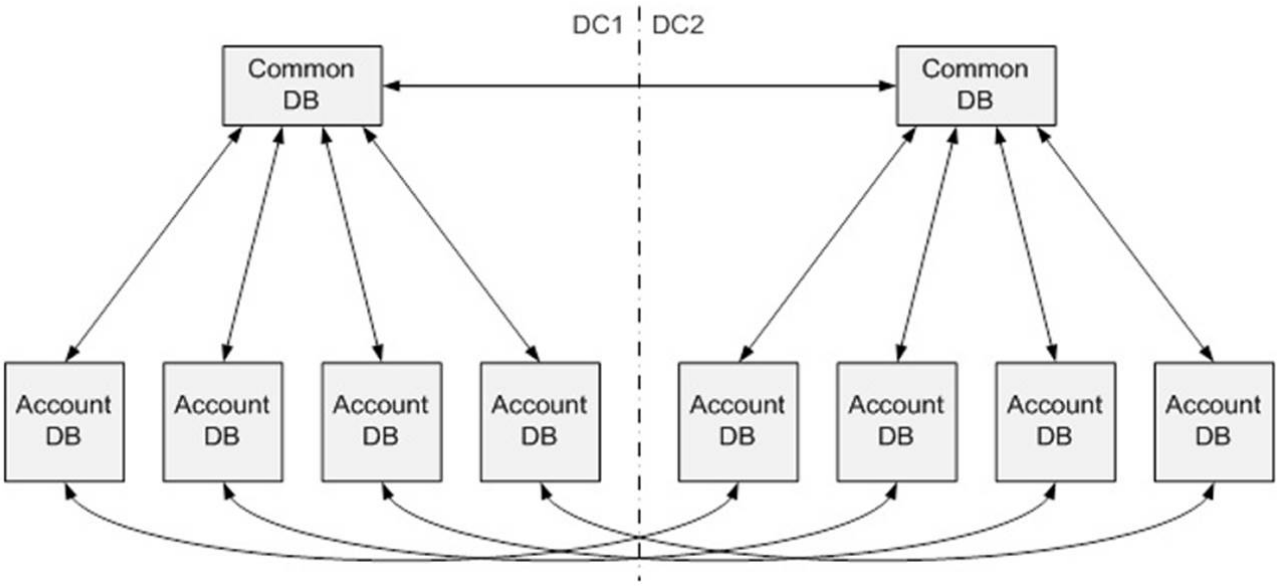


Fig 1.2.3.

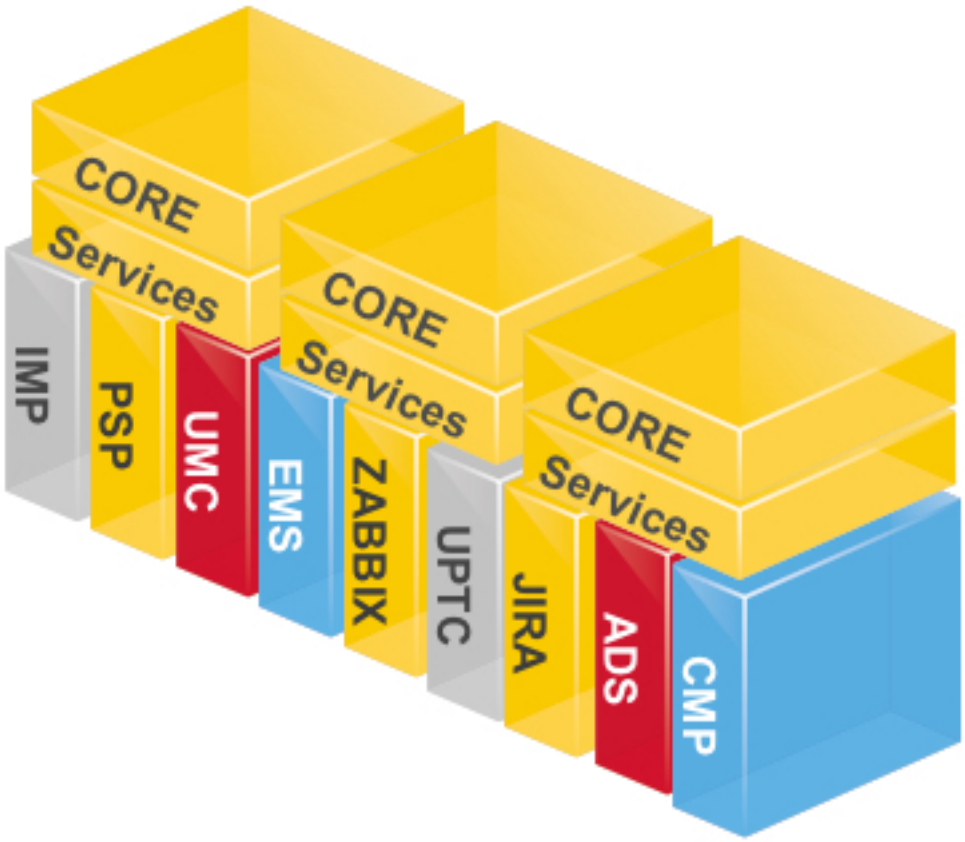


Fig. 1.2.4

All these services are part of different business processes, which are established in engineering and operations to run the core services in the non-interruptible mode and provide five nines as a level of availability. Some of the services are listed below:

- IMP – incident Management Portal
- PSP – Production Support
- EMS – Event Management System
- Zabbix – runtime metrics collection system
- JIRA – Ticketing system
- ADS – Automatic Deployment System
- CMP – Change Management Portal

Currently all the elements of the toolset mentioned above are implemented as an independent solution, by separate teams, using different technologies. The level of integration should be considered as "loose-coupled." Often, developers of each toolset operate in "silo mode" and the data about other components gets replicated. It is often very tiresome to reconcile information among multiple systems.

1.3. Directions of service expansion

In addition to existing services and business processes, companies are addressing the currently common problem across the industry: the "lead-time" of hardware provisioning and the initial high cost of ownership. Three of the main directions for moving forward, are:

- Move the whole solution to the public or managed cloud;
- Create a hybrid cloud with an ability to either deploy new services with high-speed in the cloud or make sure, in case of high demand, to be able to spill over processing to the cloud.
- Utilize containerized services, and service discovery approaches.

These three directions will add additional variables, since the asset information as well as relationship information will be dispersed across multiple locations, cloud providers, and elements of representation.

1.3.1. Complete move to public or managed cloud

This approach is trending today. There are obvious benefits of complete decoupling from hardware and Data Center related problems and operation management. This is quite convenient for companies with small Data Center footprints and startups. However, for companies that already have service(s) to run and multi-Data Center presence spread across multiple continents - this is a quite difficult and cumbersome process, which requires time for regrouping and restructuring of operations. These processes cannot be accomplished instantly and require tremendous planning. The cost of the move and real profitability requires precise calculations. Consideration of uninterruptable service to the customer is a corner stone problem.

1.3.2. Public/Hybrid cloud expansion

One of the standard problems for most of the private cloud is the balance of growth and managed cost of ownership. Each public cloud offers the ability to expand capacity if necessary. It is a swift expansion mechanism, but we still have to manage transparency of service management and control. There are multiple public cloud offerings today. The biggest services are AWS services, GCP (Google Cloud Platform), Azure from Microsoft, Oracle, RackSpace, Computer Associates, Dell, and myriads of smaller companies, which are also trying to carve out a piece of the public cloud market. The offerings include both the standard set of Compute, Network, and Storage combinations as well as

expanding on the “Something As A Service” to monetize the “bread and butter” of the each company’s core business. To optimize the cost of ownership, it is best for the company to be neutral to specific offerings of each of individual cloud. This simplifies switching from one cloud provider to another, or utilizing multiple cloud providers for cost and point of presence optimization. The prices and combinations of services are constantly changing, since each cloud provider is still looking for an opportunity to expand their customer base. As a result it means that we have to have:

- A layer of abstraction for dealing with the differences and nuances of each individual cloud. This could be possible with a few open source tools like “terraform.”
- To have an orchestration layer capable of management across multiple geo-locations and across different cloud providers. It could be even so dynamic that it could be based on the instant cost and offerings in the mode of auction and bets.
- We need to have a CMDB that will be capable of maintaining all information about a services current status, configuration, and distribution as well as across all of the things mentioned above.

1.3.3. Micro-services, containers and service discovery

One of the directions for better leveraging available resources is containerization and utilization of the standard orchestration mechanisms. It goes along with the typical stack that includes using DNS as a service (DNSAAS), Load-Balancer As A Service (LBAAS), and service discovery mechanism on top of it. It is important that LBAAS is applicable for both types of traffic:

- Internal (East-West load-balancing)
- External (North-South load-balancing), with possible federation.

One of the challenges today is that in the world of DevOps different orchestration mechanisms are more suitable for different tasks. As a result internally we are facing a necessity to support different types of orchestration.
In Fig. 1.3.2.1 the current situation in production and different technological stacks is reflected.

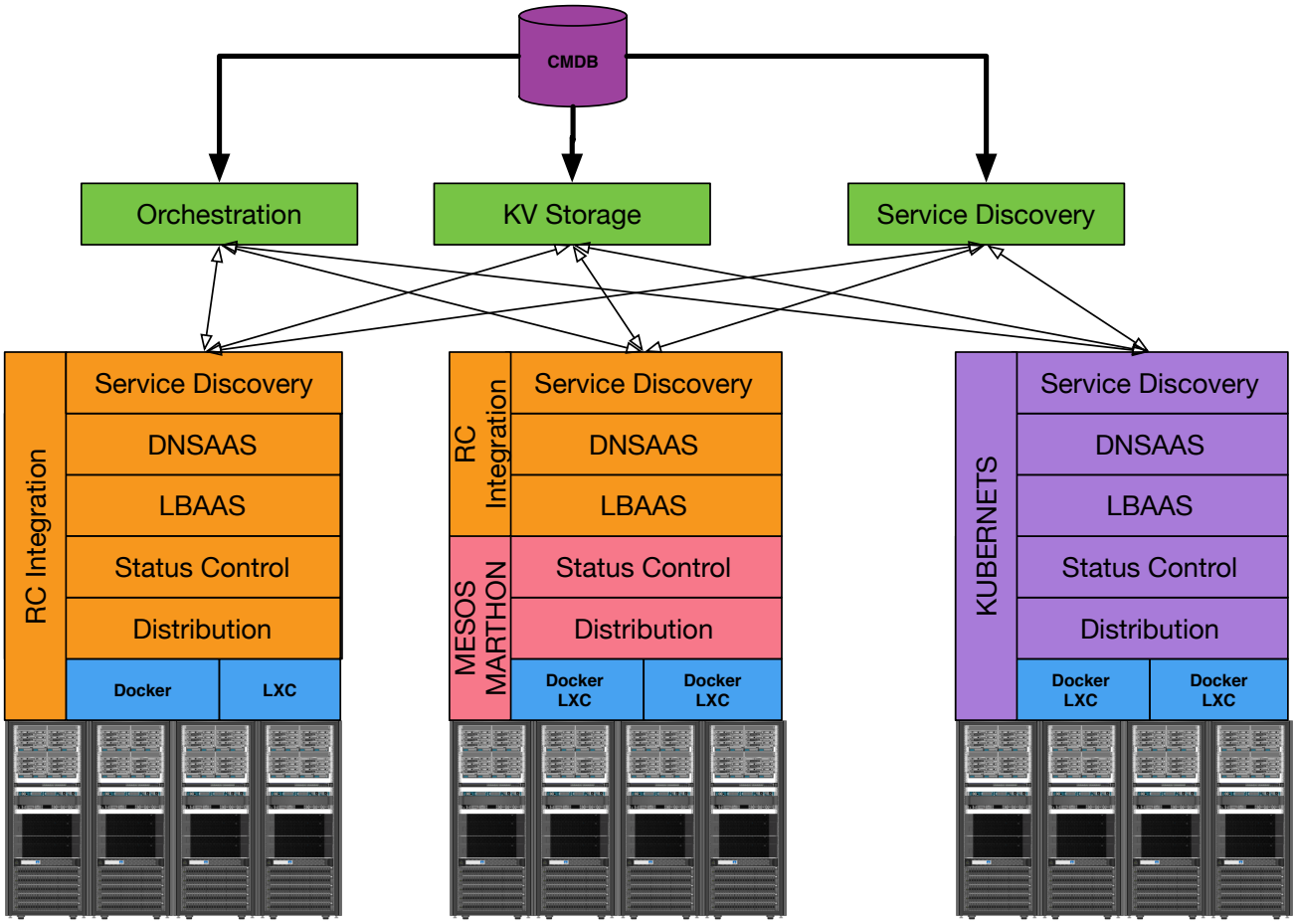


Fig. 1.3.2.1

One of the important problems, which we must solve, is “how to guarantee the same code across multiple geo-presences and clouds.”
Our approach is to leverage both described above possibilities. In particular we are currently trying to expand our presence into a few different clouds and leverage “Docker Containers” as a building block of all the services across internal and public clouds.

2. Problems

There were multiple challenges, which, we faced at the beginning of our journey two years ago:

- All the orchestration tools were operating and had their own data model. Since some of the tools were an open-source or commercial product and other tools were home grown, the integration of data/service/event models was challenging and did not fit seamlessly.
- The Complex Event Management System did not exist in the environment.
- No multi-tenancy for teams, users, and services.
- Not all sources of information were covered.
- Topology of the system was not formalized and did not cover physical and logical dependency, as a result there was no possibility to automatically conduct root cause analyses.

- The tools team was a bottle-neck, since it was supposed to handle all monitoring demands from the whole development and operations team.
- There was no simple integration between services and business processes.

Some of the most salient problems is described below.

2.1. Long-winded alarms

There wasn't awareness of logical and physical relationships. As a result of a failure of the physical host which ran multiple VMs, we got thousands of alerts about the failure of each service on VM, each VM failure, and failures of the services on the physical host, etc.

2.2. Correlation and false-positive alert invalidation

Running monitoring for logically different services and business processes, which are quite often dependent on each other and on some shared components of infrastructure. Without an advanced CEP system and integration of it with CMP (Change Management Portal) and without knowledge of topological relationships, we were getting many alerts during the beginning and ending of the maintenance window.

2.3. No single point of truth – high cost "truth" reconciliation

Different subsystems operate in the space of their own knowledge domain. As a result models in the different systems were different, inconsistent, and not reconciled. All attempts at fixing those problems were not successful since they were trying to find a solution locally and as result it brought an even higher level of complexity for the next iteration. Since some of the services were home grown and some of them were provided by a third party as an open-source product or as a commercial product, and cost of development was very high. Without proper dev-ops practice it became a very difficult and almost impossible goal to achieve.

2.4. Absence of Self-service - High-level cost of monitoring

As a service company we have the typical ratio between engineering developers and the operations team: with 1,000 developers and more than 100 engineering teams, with four annual releases, creating more than 400 components, there was just one monitoring team which consisted of seven people. With this ratio since all requests were supposed to come through this one team, it was difficult to provide an adequate service. Almost all efforts were directed to supporting the release cycle.

2.5. Service Management

This is probably one of the trickiest parts of service management. On one side, we wish to have a universal mechanism for managing of services, on another it requires whenever possible to unify most of the services or at least some major parts of it. To achieve this, the system's design should be done either in the top-down approach, when all components and service are deliberately thought through, or when we are leveraging some framework like PAAS and then we can ride on top of PAAS orchestration. Unfortunately, in our case, we have many services that were developed over the period,

not including “legacy” components, that are just using some old technologies. Also, the nature of the telephone service dictates its own logic. As a result, there are not any unified service management approaches, it makes service releases and changes very difficult, and it brings extra strain to company resources.

3. Toolset redesign – technology stack approaches

3.1. CMDB – single point of “truth”

The core part of the new toolset is CMDB.

The CMDB can significantly improve the management and delivery of an organization’s service offerings. We included the service details and relationships stored in CMDB, and it enabled us to analyze data on a service basis to perform important service management activities. Examples of these activities include:

- Timely, accurate service impact assessments for proposed changes.
- Appropriate incident response and escalation from the perspective of the service offering.
- Relevant resource focus and attention to fixing the problems that present the greatest urgency and impact to the business.
- Intuitive and effective service level monitoring and reporting of customer service commitments (service level agreements [SLAs]), related internal or external supplier partner agreements (operating level agreements [OLAs], and underpinning contracts [UCs]).
- Improved risk analysis, which will reduce potential critical failures and problems — especially when assessing the impact of changes

These activities could be accomplished only if we get multiple views through configuration structures (service configuration structures and infrastructure configuration structures) that will link our services to our underlying infrastructure. The CMDB is an excellent repository for holding the break down of the service offerings and to show the links of the underlying infrastructure as well as related internal or external supplier chain. The practice of Business Service Management (BSM) creates a bridge between service assets and business services provided at a higher level.

Equipped with this knowledge, we can identify the type of relevant management information we will require to meet the service commitments that have been made across various service types. Using this information we can determine appropriate CMDB structuring, levels, relationships, and attributes. Some organization may offer different types of services that will demand very different measurement and management characteristics. For example, some organization may provide services that are categorized as business service offerings. Ready for direct consumption, these services might include an end-to-end service, such as a server provisioning that comes complete with network connectivity, and access approved business solutions.
As a result when the different elements of the toolset are completed we will get them in following four layers as depicted on Fig. 3

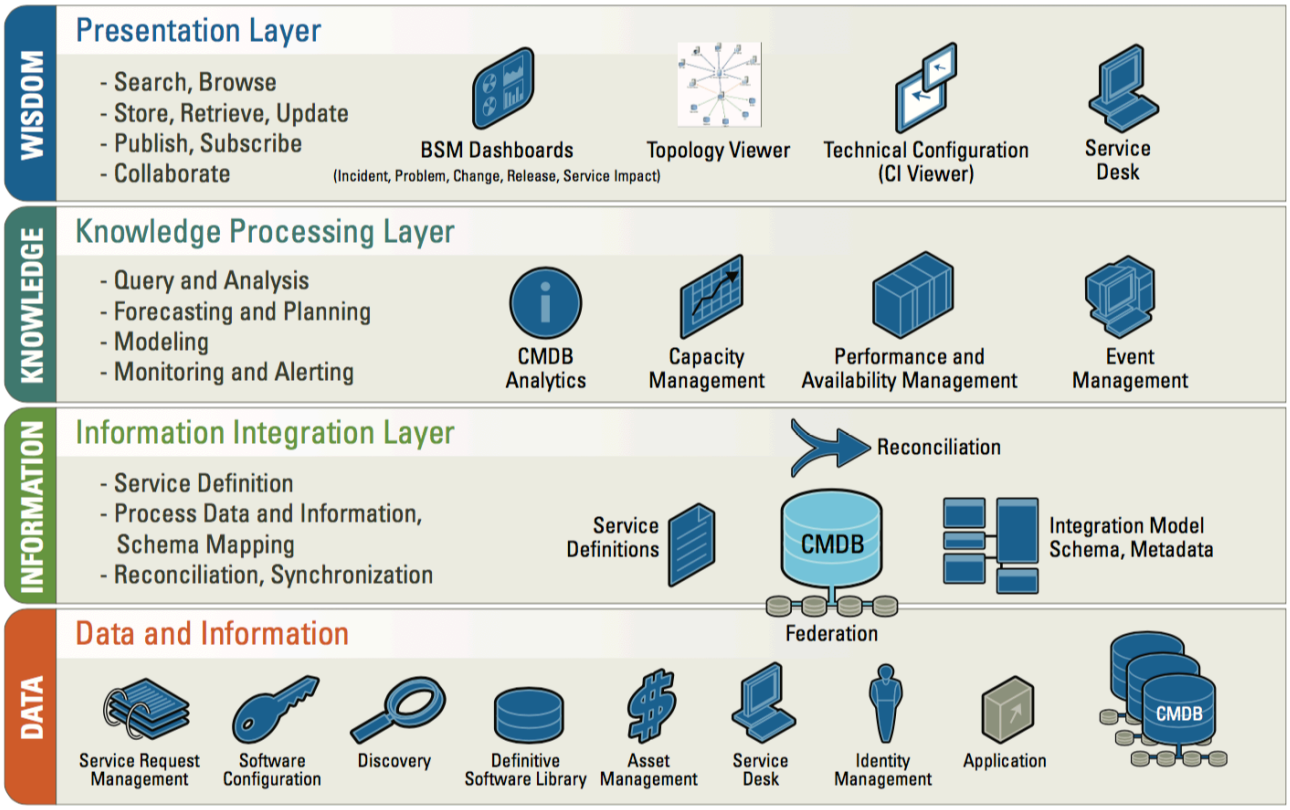


Fig. 3

3.2. ITIL V3 and the Service Management Lifecycle

ITIL V3 introduces the service management lifecycle approach, illustrated in Fig. 3.1. Keep this lifecycle approach in mind as you build your CMDB, and remember to view the CMDB in terms of how it can improve business services.

Service management begins with service strategy. The goal of service strategy is to design, develop, and implement service management as both an organizational capability and a strategic asset.

Next is service design, which is focused on ensuring that IT services offered to the business fulfill the objectives of the business. Service design is followed by service transition, in which the focus is on creating a smooth ride from design and development to operation.

Service operation then strives to effectively manage operational priorities, such as the availability of IT services provided to the business; optimize the use of existing infrastructure; resolve issues; and control demand for services.

Improving the quality of existing IT services is the heart of the value delivered by continual service improvement (CSI). While the other four parts of the service management lifecycle are best performed sequentially, CSI is actually most effective when it is embedded as part of each of these lifecycle stages.

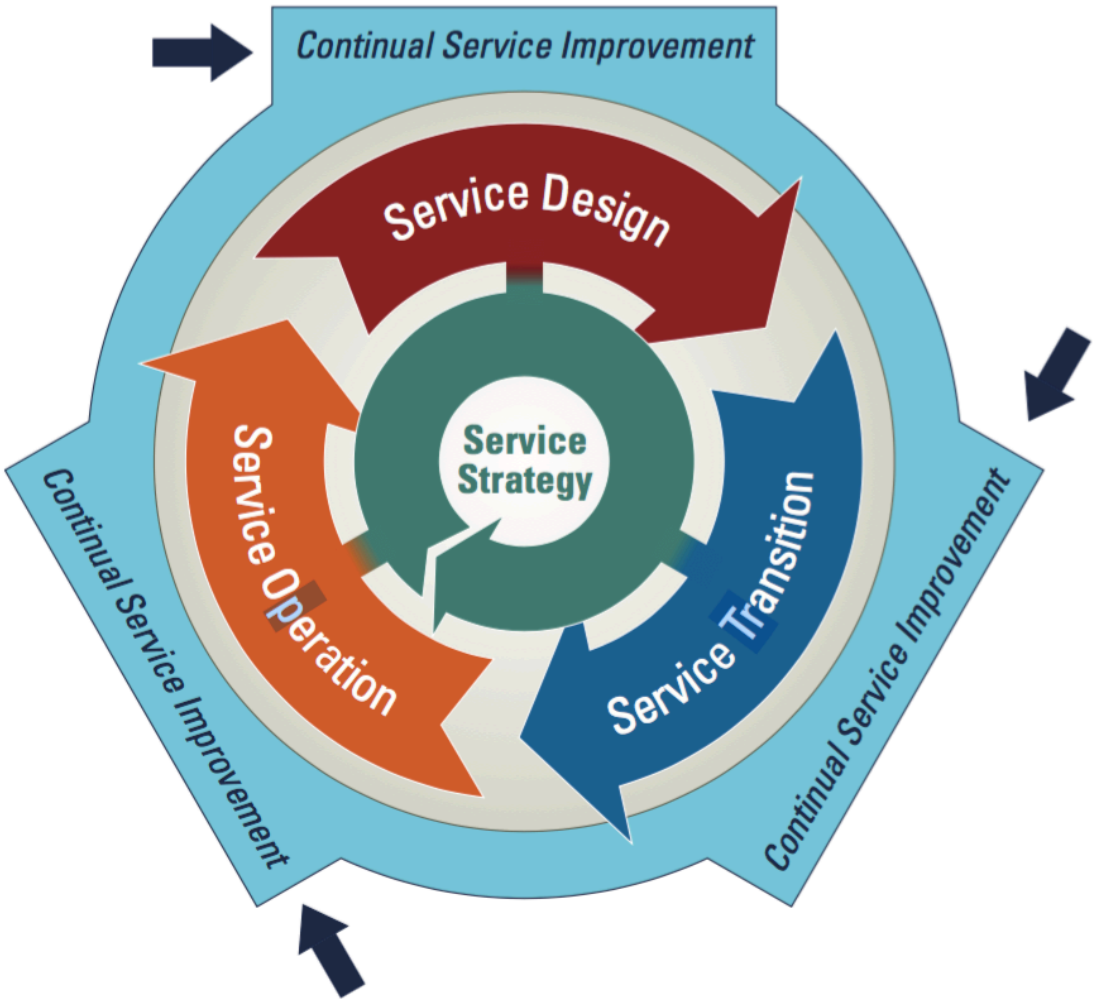


Fig. 3.1

3.3. Templates, instances and relationship

The CMDB service model is depicted in Fig, 3.3.1.

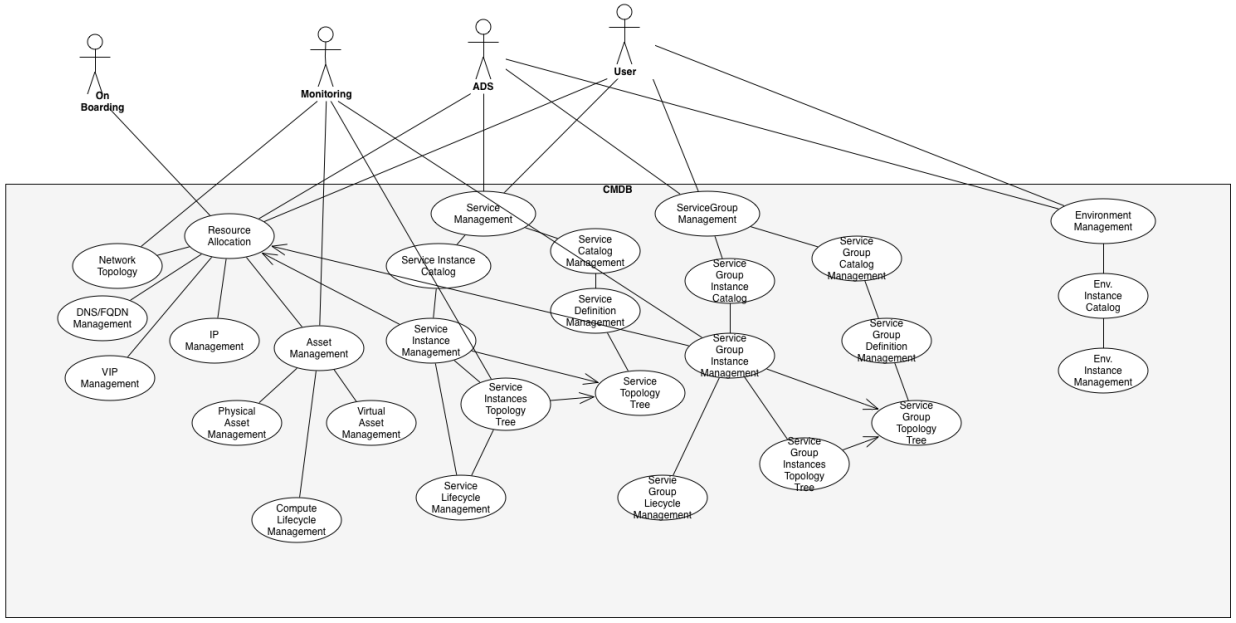
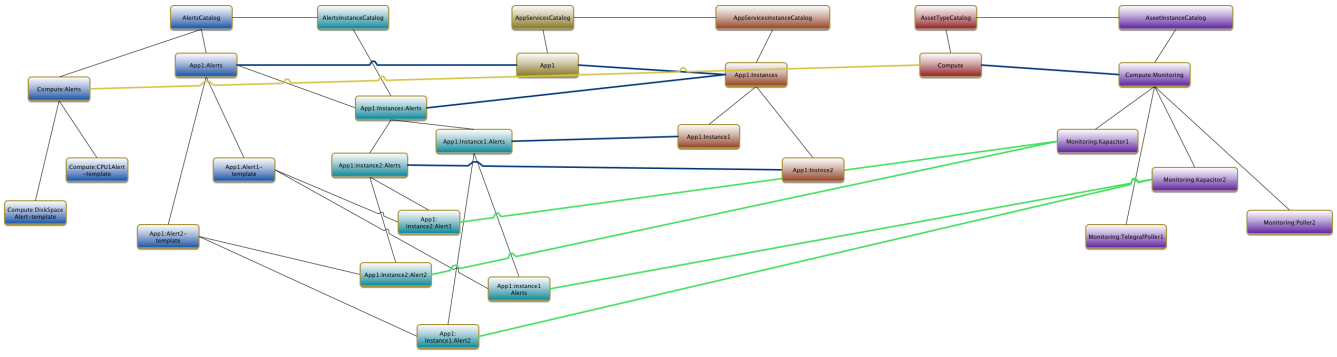


Fig.3.3.1

The primary concept that we'll be using is the concept of relationship between template and instance of this template. It will be applicable to any elements of relationship in CMDB. Conceptually we have a parameterized description of Hardware, Application, Service, Alerts, Dashboards, Correlation rules etc, and then when the service or application is initiated we will track the relationship between the template and instances of this template or version of it. This approach allows us to track down different resources and elements related to one or another version of application, and as result perform "garbage collection" of other resources that are dispersed across other different subsystems. In Fig. 3.3.2. is an example of the relationship between the application template, alerts templates (which are associated with this application template), instances of the application, and instances of the alerts that distributed across multiple time-series analyzers.



Fig, 3.3.2

3.4. Users, Consumers and Actors

There are different groups of users that will be consuming different types of information:

- NOC/Sysops/Site Reliability Engineering – real-time monitoring with key business metrics mainly aimed to the minimize time to detect and time to fix. Usually this group is looking for very granular data (5s – 15s), and the standard 1min interval usually works as well

- Upper/Medium level management – "real-time," usually this group is interested in very aggregated pictures, for the whole services or for the some specific subservices. Granularity within the range 1-5 min usually brings needed level of satisfaction
- Business analytics, capacity planning – "non real-time," difficult to predict but mainly could be used for trend detection purposes, etc. Granularity could be in the range of 5+ min. But might require high granularity of historical data.
- Developers - this group is presenting the highest volume of the requirements, data collection, and historical data.
- Deployment System – has a very specific set of requirements and with contemporary trends towards SOA and micro-SOA deployments required very particular set of tools, which will be able to verify whether that particular code deployment was successful.

Below are a few comments about nature of these consumers and their relationship with the Monitoring Team, which is going to serve all the different demands coming from consumers.

- NOC – usually not a big team, which mainly focuses on Key Business Metrics. For them the set of tools that is going to detect some anomaly in a system or site's behavior is typically important. They usually should have a set of tools that will allow them to quickly remediate a problem, such as increase capacity or fail-over switching, etc. There are many important things that should be taken care for them:
    - Root-cause priority view
    - Filter of dependency
    - Event/alert de-duping
    - Early problem detection
    - Maintenance window invalidation
    - Escalation
    - Automatic remediation of the minor problem
    - Anomaly detection
- Sysops –is usually an overloaded term and stands for everybody who is responsible for infrastructure components and system administration. It usually includes the storage team, networking team, DBAs, etc. Their view and collected metrics may vary depending on the team's specific goal Typically, sysops need to see the deviation of the stand-alone server versus the rest of the pool, or another server or system. Sometimes it could even be the situation, depending on the vendor, when only vendor specific monitoring tools are available. And it is necessary to do an additional integration for a holistic view. Their volume of request for monitoring a team is moderate, in respect of non-traditional views. Real-time view with very high-level granularity.
- Site-Reliability Engineering – closes the gap between the infrastructure and the application world. Their set of requirements may go in a very broad spectrum that covers: dependency between application and infrastructure, log analyses, logical pool dependency and application's remediation, and problem detection. Usually this is the team that supports the roll-out of the new code in production and deals directly or indirectly with deployment systems. This is the team that is deals with an application's behavioral anomalies, so they need the set of tools that will be able to help them to detect and compare current behavior with some past performance. Real-time view.
- Management – In most cases there is a requirement to see a very aggregated view of application, services, and infrastructure were the status mainly reflects with abbreviation of Green, Yellow, Red – on the top level and capability to drill down if necessary. View should be close to real-time or real-time.

- Business Analytics, Capacity planning – usually these categories require a lot of historical data, with different precision, and not related to real-time.
- Developers – this category is capable of generating the highest level of demand for adding and modification of existing monitoring views. Granularity of monitoring may vary as well as the data retention time. May require the mechanisms for thresholds of different parameters, and it could be very verbose. But the same group of developers will consume these events, without affecting other groups of consumers. The most convenient model for them is self-service, when 90% percent of required operations could be conducted via self-service portal or configuration.

3.5. Monitoring redesign

In Fig 3.5 presents the principal sets of generic components for a generic "monitoring" system. The picture itself is very much self-explanatory, but for consistency the item's description is presented below:

- Telemetry collector(s) – is a scalable component that collects and aggregates monitoring information of any type: metrics, events, traps, etc; should provide local redundancy and transport should be able to provide "best effort" on delivery. Still some type of events may require "guarantee delivery" transport;
- Telemetry Time Series DB – main storage where all delivered metrics and/or events are stored with original granularity; It should be capable to support real-time query of historical data, very important feature for anomaly detection functionalities; The rules of aggregation should have very tight information stored in the CMDB for generation of collection lists as well as invalidation of the nodes/services in accordance to the Compute and Application Life Cycle.
- Telemetry Rule Engine – is the system that is capable of performing basic temporal and spatial aggregations, threshold calculation, data normalization etc.;
- Rule DB – the database where all possible rules for Telemetry Rule Engine are expressed; we'll keep, for now, out of the discussion how it will be populated.
- CMDB – the "magical" system that contains components and services topology information as well as the status of those elements.
- Log Management Subsystem – is components where the logs from all possible sources are collected; It could/should be the source of events and metrics for the further processing;
- Consumers – some of the most prominent, but not all:
    - CEP – Complex Event Processing – is described in separate document and performing operations in the event space.
    - Auto-remediation system – has capabilities to address automatically simple use cases on the system/component level without escalation.
    - Ticketing system – is the way to track the certain type of alerts/events and their expiration.
    - Escalation – the system, which will visualize the events metrics that should be escalated.
    - Dashboards – the way to present different slices of data.

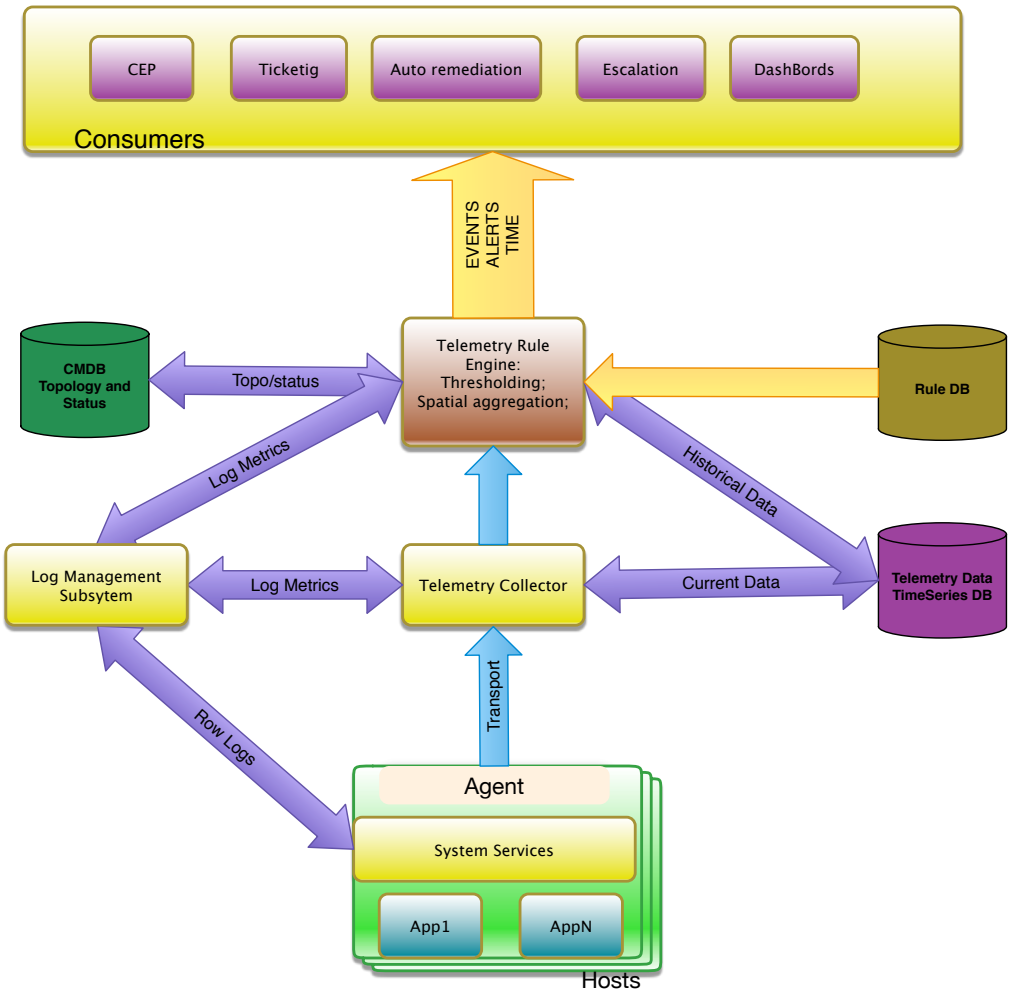

CEP
Ticketig
Auto remediation
Escalation
DashBords
Consumers
EVENTS
ALERTS
TIME
CMDB
Topology and
Status
Topo/status
Telemetry Rule
Engine:
Thresholding;
Spatial aggregation;
Rule DB
Log Metrics
Historical Data
Log Management
Subsytem
Log Metrics
Telemetry Collector
Current Data
Telemetry Data
TimeSeries DB
Row Logs
Transport
Agent
System Services
App1
AppN
Hosts

Fig. 3.5

3.6. Developers self-service

3.6.1. Monitoring as a code

The main idea here is to minimize the amount of actions and number of involved people out of the scope of the project. For all server side components as soon as there is need of emitting any new metric(s) from the component(s) the following actions should happen:

- Developer defining name of the metric, making sure it is unique in the scope of this component or whatever entity is used as a parent for this measurement;
- Developer developing the script which will be collecting data for this metric; This script will be invoked by agent frame work in correspondence of the defined time interval;
- Code is deployed on the server in correspondence with existing processes.
- After deployment, the agent's framework will detect the new monitoring script and will start invocation of that script in accordance with scheduler;
- Metrics will be collected and will be sent to the telemetry collector as part of the whole entity;
- As soon as data becomes available in the time-series DB, the developer is able to go to the monitoring portal and start observing charts of individual metrics or apply any aggregation rules to those metrics, define thresholds, action/reactions etc.

This scenario requires 'zero' involvement of the monitoring team.

3.6.2. Deployment as a code

The important part is to offer to developers the ability to describe their applications and services in such a way as to allow them to automatically deploy and build the whole application stack. The manifest is created by developers and used by automatic deployment systems as a blueprint, which describes how the service will be used, what are dependencies of the application, how it will be monitored, etc. This system should able to efficiently share and utilize underlying resources. It delivers robust orchestration of heterogeneous stacks and environments - containers alongside non-containers, stateful and stateless services, all through a single application blueprint.
This includes:

- Multiple hosts
- Placement control
- Network orchestration
- Placement
- Affinity/anti-affinity
- High availability
- Scaling
- Load balancing
- Rolling upgrades

3.7. Combining of logical and physical topology

Fig. 3.7.1 depicts the model which is describing a relationship between hardware related elements of infrastructure and a relationship between these elements. Fig. 3.7.2. shows a logical relationship between applications and services. We split these into different figures because otherwise it would be too complicated to visualize. The reality is that these are merely parts of the same unified topological relationship that is populated and maintained inside CMDB. This topology is traversable in any direction so the consumer of this system knows or is able to calculate at any moment the relationship between instance of application service and VM-> Physical Host -> Switch -> Rack relationship.

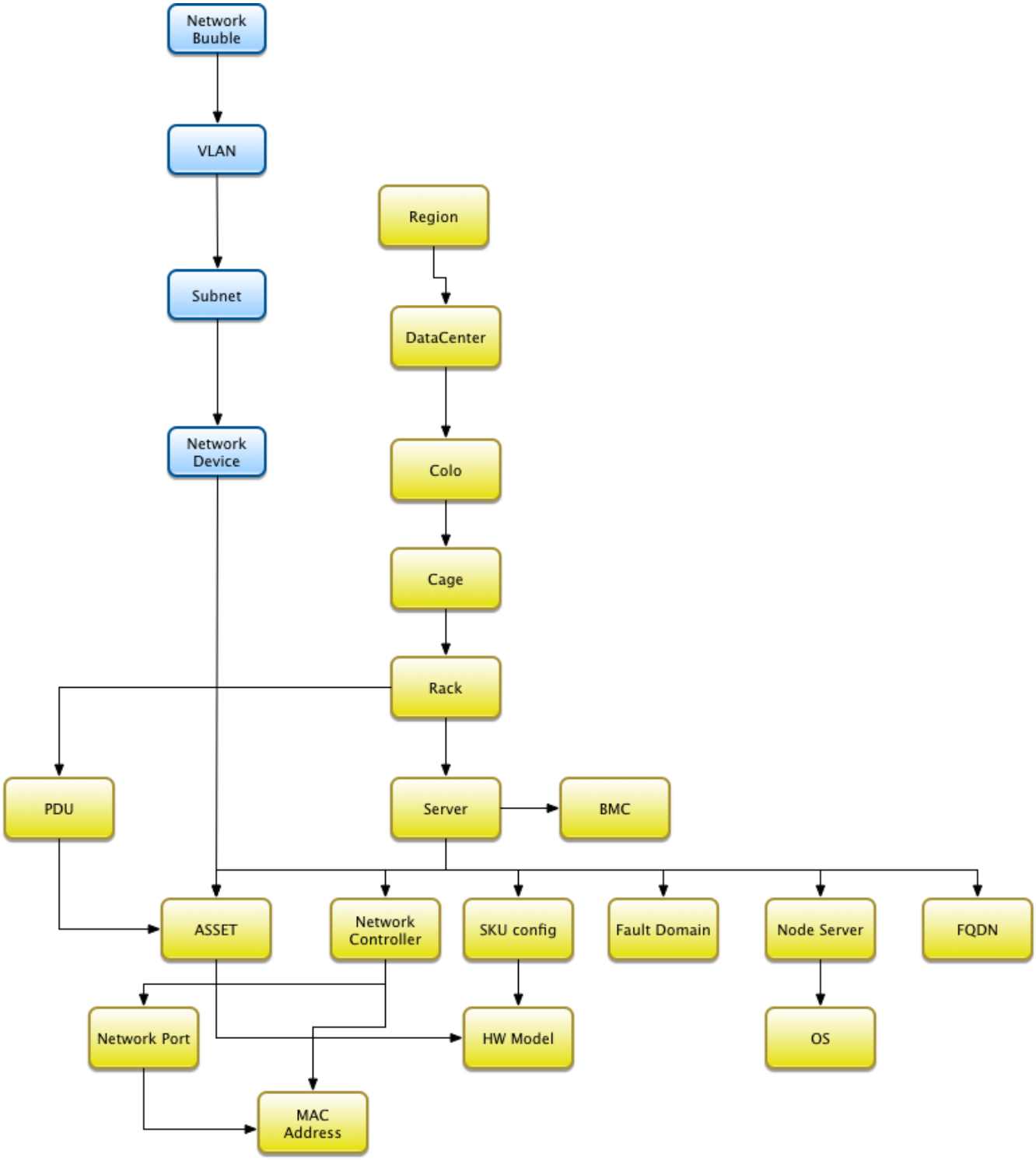


Fig. 3.7.1

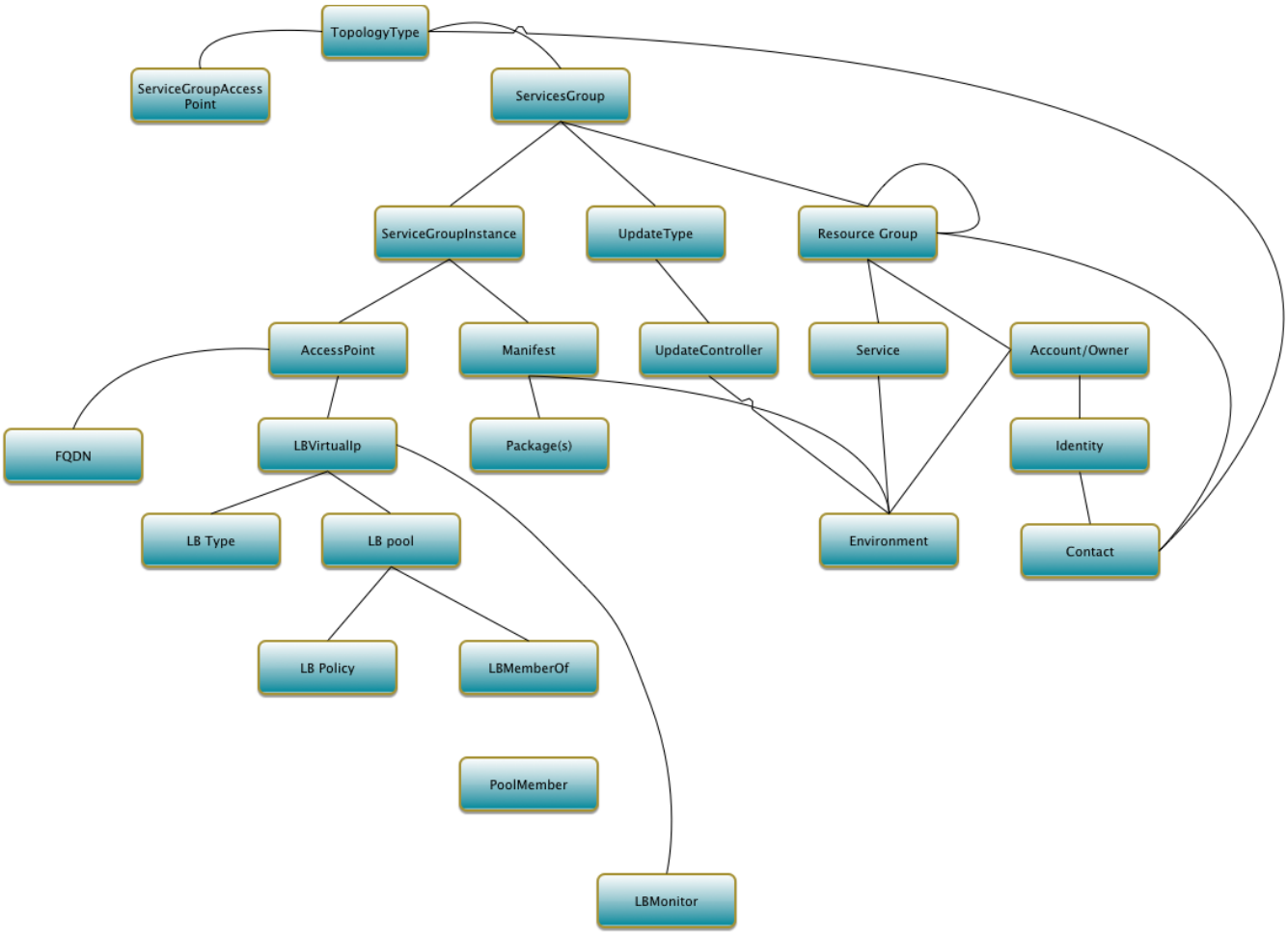


Fig. 3.7.2

3.8. Automatic Data Discovery Model – ADDM

Building dynamic application maps has many challenges; but for running services with high reliability it is necessary to dynamically verify and capture accurate inventory of physical, virtual, and logical infrastructure and application assets, which requires the ability to detect domain knowledge. It should be supporting various virtualization platforms and technologies, like vSphere, Open Stack, LXD, and contemporary container orchestration frameworks like Kubernetes, Docker Swarm, DCOS, as well as it should be capable of support public cloud infrastructures like AWS, GCP, etc. Also it should be capturing application services. It is capable of dynamically binding individual application and services configurations into the logical groups that forms a specific multi-tier application.

3.9. Compute and Application Lifecycle

Status and state describe the current phase of the lifecycle of an asset.

3.9.1. Status

The lifecycle (from birth to death) of an asset is described in terms of its status. The possible status values are fixed. All proposed status values are listed below, with some descriptions given, primarily to indicate their meanings for a server that could be physical, virtual, or container, and any combination thereof. A non-server asset type such as a configuration may have its own transition diagram.

Status	Description
Incomplete	Host/VM/CT not yet ready for use.
New	Host had been racked and waiting for completion initialization;
Unallocated/Cache	Host has completed initialization process and is ready for allocation
Provisioning (HW)	Host has started provisioning process but has not yet completed it
Allocated	This asset is in what should be considered a production state
PREP (Application provisioning)	Means that application is in process of installation and configuration
LIVE	Means – in service; live customers are on;
Remediation	Means application/host is in remediation status
Cancelled	Asset is no longer needed and is awaiting decommissioning
Decommissioned	Asset has completed the outtake process and can no longer be managed
Debug	Indicates that host is in debug mode and metrics/alerts from this host, could be ignored.
Maintenance	Asset is undergoing some kind of maintenance and should not be considered for production use

The status transition should not generally happen by hand. Automated processes should drive status changes, not people.

CLM statuses:

- Not exist/Planned (item does not exist yet but preparations to purchase or reserve resources have begun, etc.)
- Exist/Ready for installation (item exists, resources reserved, item is ready for installation)
- Installed/Ready for setup (item is physically installed)
- Setup/Ready to use (item is set up, accessible)
- In use
    - Active
    - Standby
- Maintenance
- Decommissioned

3.9.2. State

While the status of an asset describes where it is in a discrete lifecycle, the state describes a lifecycle specific to a status. For example, a server that is in maintenance may have a state of HARDWARE_PROBLEM or HARDWARE_UPGRADE. Also note that those states are not appropriate for HEALTHY (non-maintenance) assets, and so these states are restricted to assets with a status of Maintenance.

A state can be either a system state or a non-system state. System states can not be modified or destroyed. Non-system states can be modified and destroyed. Via the API you can only create non-system states, although support for adding system states may be added in the future. A state can be bound to a status (such as the case of HARDWARE_PROBLEM), or can be used with any status (such as the case of RUNNING). The out-of-the-box available states are described below.

3.9.3. Application Lifecycle management

In this section we present an idea that during normal operation some application may have different patterns of behavior that are important to know since otherwise it may lead to some false positive alarms.

Status	**SubStatus**	**Description**
Started	N/A	Application/Service is started.
Stopped	N/A	Application/Service is Stopped.
InService		Means that Application/Service is servicing live traffic
OutOfService/Prep		Means that Application/Service is not servicing live traffic and could be in some preparation mode

InService /OutOfService/Prep	Init	Application/Service is performing some initialization
InService /OutOfService/Prep	Runtime1	Application/Service is normal operational mode1
InService /OutOfService/Prep	Runtime2	Application/Service is normal operational mode2
InService /OutOfService/Prep	Runtime3	Application/Service is normal operational mode3
InService /OutOfService/Prep	Sync-up	Application/Service is in sync-up mode

For the most applications, the ARLM is very primitive and comprised of just Start/ OutOfService/ InService(Runtime1)/ OutOfService/ Stop. But there are a few examples of more complicated behavior. For instance, we can take a look on the MongoDB cluster lifecycle:
Start – MongoDB instance is started
Init – MongoDB is loading data.
InService(Runtime1)- Servicing Rad/Write queries
OutOfService|InService(Runtime2) – may occur if we are asserting a new node into replica set and as a result the node will start consuming a lot of IOPS and incoming bandwidth utilization will go up.
InService(Runtime3) maybe depicting the fact that this node not only servicing the normal volume of Read/Write requests but also is a source for replication to the new member of the replica set.

3.10. CEP and Event management for tacit alerting

3.10.1. Type of processing and definitions

3.10.1.1. Filtering and thresholding

The query subscribes to one stream, evaluates a specified logical condition based on event attributes, and, if the condition is true, publishes the event to the destination stream. For example, an application monitoring a stream of purchase orders may filter out all orders where the condition is **Priority != 'High' and CPU_system > 200**.

3.10.1.2. Data Caching

The cache typically stores two kinds of data:

- Recent events from one or more streams

Recent events are typically stored in windows. A window is an object, similar to an in-memory database table. However, a window can manage its state automatically, by keeping and evicting certain events according to its policy. For example, a window policy might specify: KEEP 1000 ROWS PER Id. This window maintains 1000 rows for each ID value, and expires old rows, as necessary.

- Data from one or more name space tables

Just as streaming events can be cached in memory, it often makes sense to cache data from other sources like: relational or non-relational database, so that different kinds of operations may be performed on this data more efficiently. This cache is typically managed according to the Least Recently Used (LRU) algorithm, or by explicit invalidation.
Note that, although we are describing an in-memory cache here, many applications require this cache to be persistent. This means that, if a machine that hosts the CEP engine fails, the data kept in windows is not lost. This functionality is even more important when the window holds not just the last few seconds', but also minutes', hours', days', and even weeks' worth of events.

3.10.1.3. CMDB Integration

Usually CMDB is a generic name for Configuration Management Data Base and has a different implementation and may be as a single or multiple instances. Usually it reflects following state and relationship:

- CLM – compute life cycle management;
- ALM – application life cycle management
- Systems' Topology and relationship;
- Applications' Topology and relationship
- Application's dependency;

This information could be used for invalidation and/or verification of the events, or generate the missing event.

3.10.1.4. Aggregation over time windows (Temporal Aggregation)

This type of processing uses the stored values to compute various statistics. A typical example here would involve computing a running average over a sliding window.
It comes in quite a few flavors, differing along the following dimensions:

- The kinds of aggregators computed

These include running averages, sums, counts, minimum, maximum, standard deviation, user-defined aggregators, and so on.

- The kinds of windows used

These include time-based and count-based windows, sliding and jumping (tumbling) windows, windows that keep the specified number of largest or smallest elements, and so on.

- Output frequency: continuous vs. periodic

In the case of continuous output (also called "tick-by-tick" output) each incoming event updates the calculated expression and an output event is produced.

3.10.1.5. Spatial Aggregation - Correlation (joins)

Joining Multiple Event Streams
While simple applications often look just at one stream at a time, most advanced applications must look at and correlate events across multiple streams. A join in a CEP application shares many characteristics with a join in SQL.

In CEP, a join between two data streams necessarily involves one or more windows. Streams do not store events, but pass events to, from, and between queries. To perform a join, it is necessary to store some events in memory, to wait for events on the corresponding stream. This is what a window does.

3.10.1.6. State Machines

State machines are used in a wide variety of applications and systems, where complex behavior and processes need to be modeled and tracked. A simple finite state machine (FSM) defines a set of states for a process, together with events that define transitions from one state to another.
A few important considerations when designing a finite state machine in a CEP environment are:

- The metadata's location
- Tracking multiple processes
- Coping with imperfections and eventual consistency

While in an ideal world all processes would always be in a valid state, events would never be lost, and no illegal transitions would occur. However, the real world is far more complex. A production finite state machine is usually designed to cope with and recover from these failures.

3.10.1.7. Hierarchical Events

The following operations may be performed on hierarchical events:

- Decomposing hierarchical events. A complex, hierarchical event may need to be decomposed into simpler events.
- Correlation across hierarchical events.
- Composing a hierarchical event.

3.10.1.8. Event Pattern matching

Suppose we want to be notified if, within a 10-minute interval, event A occurs, followed by event B, followed by either event C or D, followed by the absence of event E, with all events relating to each other in some way. While such an event pattern can be tracked with a combination of inner and outer joins, it is often desirable to have a more direct way of expressing such time-based relationships.

Most interesting event patterns involve any number of relationships among events:

- A followed by B - Event B occurs after event A.
- A and B - Both events A and B occur, in either order.
- A or B - Either A or B (or both) occur.
- Not A - Event A does not occur.
-

Some of the most interesting definition involves "negative" events, in which the pattern matching criteria are met when a specified event does not occur within the specified time interval. For example, when tracking a process based on requests and responses, it may be important to know when a response does not occur within a specified time period from the request, or when a response occurs without first being preceded by a request.

3.10.1.9. Events DE-duping and/or event Rate changes

It could be considered as some version of filtering; despite in practice it has very significant presence. While receiving the same event a system can apply time based or counter based event reduction. Same event definition may go the same host or pool or any other metrics; the result of de-duping can lead to either complex event creation or change some event properties like severity or the escalation field.

3.10.1.10. Events Suppression

As well as the previous example, it could be considered a narrowed version of the filtering system, but it also has a tremendous practical value.

The idea here is that system can validate the status of the Asset/Pool/Group etc. and based on this information discard the event because it is in maintenance mode, or the application is currently executing different activity, for example query against index versus index generation or download.

3.10.1.11. Timer Based events

This is the capability of the system to associate a separate timer and generate an additional event in case of the timer's expiration. A typical example is: Maintenance window initialized and by the standard could last up to three hours; if the close of Maintenance window will not arrived before expiration a new event will be generated.

3.10.1.12. Detection of missed events

Capability to detect that event isn't coming, from a certain source. It usually falls into two categories:

- Capabilities to interpret CLM or ALM type of information, for defining that something is supposed to emit events, which are not arriving;
- Events are coming at a specific pace, and suddenly stop coming;

4. Long-terms goals and further development

4.1. Ability to detect missing data

An important ability is to detect the fact that data is supposed to be flowing into a system but is missing due to unknown reasons. It is based on the idea of comparison WISBe (Where It's Supposed to Be)

and WIRI (Where It Really Is) models. The WISBe model is coming again from CMDB, and WIRI is expected also either to be in CMDB or be a local property of a monitoring system. As a result, an event/alert could be generated, which can cause either escalation or auto-remediation action(s)

4.2. Anomaly detection mechanism based on historical behavior patterns

Anomaly detection was a very murky area but there is now clear definition. To a certain extent it goes into the area of prediction of the system's behavior. Here we are presenting the requirement of having out-of-the-box mechanisms, which will allow us to have set of tools leading to implementation of "Bollinger Band," with some variations of it. Classical "Bollinger Band" will predict or define "band" or range where an expected measurement will be based on past data points.
On top of that, because our service has a property when there is more activity versus when there is almost no activity, we wish to have an extended set of bands, based on historical behavior of the service, like following:

- An hour ago;
- A day ago;
- A week ago;
- A month ago;
- A season ago;

The depth of data collections also depends on historical definition of the calculated in the past aggregated metric.

4.3. Real time anomaly detection

The set of tools and instrumentation, which could be used for real-time value prediction based on the limited history; a typical example – "Bollinger Band"

4.4. Dynamic threshold support

Typically, the contemporary monitoring systems widely use threshold mechanisms. But this is usually related to the statically defined value expressed either as constant for the value or for the percentile (like if CPU more than 60% send alert).
In contrast, a service's behavior has multiple properties, one of each, depending on the time of the day, minute of the hour, day of the week, etc. (For example the activity of meeting creation is bursting during business hours, or auctions are usually bound to the end of the hour, etc). As a result it requires the capability to define the threshold based on the other different parameters and not only as an absolute value.

4.5. Fit for Highly-Distributed or SOA environments

- Ability to monitor the health of individual Application Servers and allow the monitoring of distributed Business Transactions;
- Automatically discover and visualize the relationship between services and the back-end tiers;

- Capability to measure and display the latency of each hop;
- Ability to identify communication problems between the tiers/services in a SOA environment;

4.6. Behavior Learning

- Continuous learning of the normal performance of the application without necessity of setting manual thresholds;
- Ability to learn different load and response time patterns for different days or times of day;
- Ability to learn “normal” deviation patterns and trigger action automatically based on what it observes;
- Capability of the system to automatically adapt to dynamic changes in capacity in a virtualized or elastic computing environment;

4.7. Reduce Mean-time-to-Resolution (MTTR) with Troubleshooting & Root Cause Diagnostics

- Ability to capture diagnostic data automatically, when a problem occurs;
- Capability to trigger the code-level diagnostic capturing for every bad session, if defined
- Should be able to distinguish one-time anomalies from consistent patterns of poor performance;
- Should have an out of box support for specific diagnostics for each of the following:
    - Slow response
    - Errors
    - Stalls
    - Deadlocks
    - Malformed code
    - Slow SQL/DB access
    - Memory Leaks & Thrash
    - Distributed service communication issues
    - Synchronization issues

5. References

1. RingCentral official web site. Available: http://www.ringcentral.com/
2. **Mescheryakov S., Shchemelinin D.** Automatic Software Deployment on Cloud Based Environments / St. Petersburg State Polytechnic University Journal. Computer Science. Telecommunications and Control Systems. – 2014. – N 1(188).
3. Zabbix official web site. Available: http://www.zabbix.com/product.php
4. **Ardulov Y., Mescheryakov S., Shchemelinin D.** Dynamic Load Balancing and Continuos Service Delivery in a Big Cloud Infrastructure, Proceedings of the 41st International Conference for Performance and Capacity by CMG, San Diego, CA, USA (2016),
5. **Ardulov Y., Mescheryakov S., Shchemelinin D.,** "Monitoring and Remediation of Cloud Services Based on 4R Approach", Proceedings of the 41st International Conference for Performance and Capacity by CMG, San Antonio, TX, USA (2015), **https://www.cmg.org/conferences/performance-capacity-2015/**
6. Dockers official web site. Available: https://www.docker.com/
7. Kubernetes official web site. Available: https://kubernetes.io/
8. LXD official web site. Available: https://linuxcontainers.org/lxd/
9. DCOS official web site. Available:https://dcos.io/
10. Market Report | Inside the Linux Container Ecosystem, SDxCentral