Mesosphere

Overview

ENTERPRISE DC/OS Updated: October 7, 2016

DC/OS is a distributed operating system based on the Apache Mesos distributed systems kernel. It enables the management of multiple machines as if they were a single computer. It automates resource management, schedules process placement, facilitates inter-process communication, and simplifies the installation and management of distributed services. Its included web interface and available command-line interface (CLI) facilitate remote management and monitoring of the cluster and its services.

To get a better feeling for what DC/OS is and what it does, jump into some of the other sections:

What is DC/OS?

DC/OS is a distributed operating system based on the Apache Mesos distributed systems kernel. It enables the management of multiple machines as if they were a single computer. It a...

Architecture

An operating system abstracts resources such as CPU, RAM, and networking and provides common services to applications. DC/OS is a distributed operating system that abstracts the re...

Features

This is an overview of the features that make DC/OS more than the sum of its parts. High Resource Utilization Mixed Workload Colocation Container Orchestration Extensible Resource ...

Components

DC/OS is comprised of many individual open source components that are precisely configured to work together. You can log into any host in the DC/OS cluster and view the currently r...

Concepts

This page contains terms and definitions for DC/OS. Admin Router The Admin Router runs on the DC/OS master servers to provide a proxy for the admin parts of the cluster. Agent node...

High Availability

This document discusses the high availability features in DC/OS and best practices for building highly available applications on DC/OS. Leader/follower architecture A common patter...

Telemetry

To continuously improve the DC/OS experience, a telemetry component is included that reports anonymous usage data to Mesosphere. This data is used to monitor the reliability of cor...

Service Discovery

There are two levels of service discovery in DC/OS. Along with every task that runs on DC/OS being provided a well known DNS name, anyone can request a well known VIP that enables ...

Roadmap

This roadmap provides a high level overview of the themes that the DC/OS project is focusing on in the near, medium and long term. Please see the design docs and JIRAs for addition...

Feature Maturity

The purpose of the feature maturity phases is to educate customers, partners, and Mesosphere field and support organizations about the maturity and quality of features. Criteria Ph...

Architecture

ENTERPRISE DC/OS Updated: September 19, 2016

An operating system abstracts resources such as CPU, RAM, and networking and provides common services to applications. DC/OS is a distributed operating system that abstracts the resources of a cluster of machines and provides common services. These common services include running processes across a number of nodes, service discovery, and package

management. This topic discusses the architecture of DC/OS and the interaction of its components.

To simplify the understanding of DC/OS, we will reuse the Linux terminology for kernel and user space. The kernel space is a protected area that is not accessible to users and involves low-level operations such as resource allocation, security, and process isolation. The user space is where the user applications and higher order services live, for example the GUI of your OS.

100,000 ft view

The DC/OS kernel space is comprised of Mesos masters and Mesos agents. The user space includes System Components such as Mesos-DNS, Distributed DNS Proxy, and services such as Marathon or Spark. The user space also includes processes that are managed by the services, for example a Marathon application.

Before we dive into the details of the interaction between different DC/OS components, let's define the terminology used.

- Master: aggregates resource offers from all agent nodes and provides them to registered frameworks.
- Scheduler: the scheduler component of a service, for example the Marathon scheduler.
- User: also known as a client, is an application either internal or external to the cluster that kicks off a process, for example a human user that submits a Marathon app spec.
- Agent: runs a discrete Mesos task on behalf of a framework. It is an agent instance registered with the Mesos master. The synonym of agent node is worker or slave node. You can have private or public agent nodes.
- Executor: launched on agent nodes to run tasks for a service.
- Task: a unit of work scheduled by a Mesos framework and executed on a Mesos agent.
- Process: a logical collection of tasks initiated by a client, for example a Marathon app or a Metronome job.

Kernel space

In DC/OS, the kernel space manages resource allocation and two-level scheduling across the cluster. The two types of processes in the kernel space are Mesos masters and agents:

- Mesos masters The `mesos-master` process orchestrates tasks that are run on Mesos agents. The Mesos Master process receives resource reports from Mesos agents and distributes those resources to registered DC/OS services, such as Marathon or Spark. When a leading Mesos master fails due to a crash or goes offline for an upgrade, a standby Mesos master automatically becomes the leader without disrupting running services. Zookeeper performs leader election.
- Mesos agents: Mesos agent nodes run discrete Mesos tasks on behalf of a framework. Private agent nodes run the deployed apps and services through a non-routable network. Public agent nodes run DC/OS apps and services in a publicly accessible network. The

`mesos-slave` process on a Mesos agent manages its local resources (CPU cores, RAM, etc.) and registers these resources with the Mesos masters. It also accepts schedule requests from the Mesos master and invokes an Executor to launch a Task via containerizers:

The Mesos containerizer provides lightweight containerization and resource isolation of executors using Linux-specific functionality such as cgroups and namespaces.

The Docker containerizer provides support for launching tasks that contain Docker images.

User space

The DC/OS user space spans systemd services and user services such as Chronos or Kafka.

Systemd services are installed and are running by default in the DC/OS cluster and include the following:

The Admin Router is an open source NGINX configuration that provides central authentication and proxy to DC/OS services.

Exhibitor automatically configures ZooKeeper during installation and provides a usable Web UI to ZooKeeper.

Mesos-DNS provides service discovery, allowing apps and services to find each other by using the domain name system (DNS).

Minuteman is the internal layer 4 load balancer.

Distributed DNS Proxy is the internal DNS dispatcher.

DC/OS Marathon, the native Marathon instance that is the 'init system' for DC/OS, starts and monitors DC/OS services.

ZooKeeper, a high-performance coordination service that manages the DC/OS services.

User services

A user service in DC/OS consists of a scheduler (responsible for scheduling tasks on behalf of a user) and an executor (running tasks on agents).

User-level applications, for example an NGINX webserver launched through Marathon.

Distributed process management

This section describes the management of processes in a DC/OS cluster, from the resource allocation to the execution of a process.

At a high level, this interaction takes place between the DC/OS components when a user launches a process. Communication occurs between the different layers, such as the user interacting with the scheduler, and within a layer, for example, a master communicating with agents.

Here is an example, using the Marathon service and a user launching a container based on

a Docker image:

The chronological interaction between the above components looks like this. Notice that Executors and Task are folded into one block since in practice this is often the case.

In detail, here are the steps:

1. Client/scheduler initialization: the client needs to know how to connect to the scheduler in order to launch a process, for example via Mesos-DNS or DC/OS CLI.
2. Mesos master sends resource offer to scheduler: the resource offers are based on cluster resources managed through agents and the DRF algorithm in Mesos master.
3. The scheduler declines resource offers because no process requests from clients are pending. As long as no clients have initiated a process, the scheduler will reject offers from the master.
4. Client initiates process launch. For example, this could be a user creating a Marathon app via the DC/OS Services tab or via the HTTP endpoint `/v2/app`.
5. Mesos master sends the resource offers. For example, `cpus(*):1; mem(*):128; ports(*):[21452-21452]`
6. If resource offer matches the requirements the scheduler has for the process, it accepts the offer and sends a `launchTask` request to the Mesos master.
7. The Mesos master directs Mesos agents to launch tasks.
8. The Mesos agent launches tasks via executor.
9. The executor reports task status to the Mesos agent.
10. The Mesos agent reports task status to the Mesos master.
11. The Mesos master reports task status to scheduler.
12. The scheduler reports process status to client.

Features

ENTERPRISE DC/OS Updated: September 19, 2016

This is an overview of the features that make DC/OS more than the sum of its parts.

- High Resource Utilization

High Resource Utilization

DC/OS makes it easy to get the most out of your compute resources.

Deciding where to run processes to best utilize cluster resources is hard, NP-hard in-fact. Deciding where to place long-running services which have changing resource requirements over time is even harder. In reality there's no single scheduler that can efficiently and effectively place all types of tasks. There's no way for a single scheduler to be infinitely configurable, universally portable, lightning fast, and easy to use – all at the same time.

DC/OS manages this problem by separating resource management from task scheduling. Mesos manages CPU, memory, disk, and GPU resources. Task placement is delegated to higher level schedulers that are more aware of their task's specific requirements and constraints. This model, known as two-level scheduling, enables multiple workloads to be colocated efficiently.

Mixed Workload Colocation

DC/OS makes it easy to run all your computing tasks on the same hardware.

For scheduling long-running services, DC/OS tightly integrates with Marathon to provide a

solid stage on which to launch microservices, web applications, or other schedulers.

For other types of work, DC/OS makes it easy to select and install from a library of industry-standard schedulers. This opens the door to running batch jobs, analytics pipelines, message queues, big data storage, and more.

For complex custom workloads, you can even write your own scheduler to optimize and precisely control the scheduling logic for specific tasks.

Container Orchestration

DC/OS provides easy-to-use container orchestration right out of the box.

Docker provides a great development experience, but trying to run Docker containers in production presents significant challenges. To overcome these challenges, DC/OS includes Marathon as a core component, giving you a production-grade, battle-hardened scheduler that is capable of orchestrating both containerized and non-containerized workloads.

With Marathon, you have the ability to reach extreme scale, scheduling tens of thousands of tasks across thousands of nodes. You can use highly configurable declarative application definitions to enforce advanced placement constraints with node, cluster, and grouping affinities.

Extensible Resource Isolation

DC/OS makes it possible to configure multiple resource isolation zones.

Not all tasks have the same requirements. Some require maximum isolation for security or performance guarantees. Others are ephemeral, public, or easily restarted. And most are somewhere in between.

The simplest isolation method is to just delegate to Docker. It's trivial to run Docker containers on DC/OS, but Docker is a bit of a blunt instrument when it comes to isolation. The Mesos containerizer is much more flexible, with multiple independently configurable isolators, and pluggable custom isolators. The Mesos containerizer can even run Docker containers without being chained to the fragility of `dockerd`.

Stateful Storage Support

DC/OS gives your services multiple persistent and ephemeral storage options.

External persistent volumes, the bread and butter of block storage, are available on many platforms. These are easy to use and reason about because they work just like legacy server disks, however, by design, they compromise speed for elasticity and replication.

Distributed file systems are a staple of cloud native applications, but they tend to require thinking about storage in new ways and are almost always slower due to network-based interaction.

Local ephemeral storage is the Mesos default for allocating temporary disk space to a

service. This is enough for many stateless or semi-stateless 12-factor and cloud native applications, but may not be good enough for stateful services.

Local persistent volumes bridge the gap and provide fast, persistent storage. If your service is replicating data already or your drives are RAID and backed up to nearline or tape drive, local volumes might give you enough fault tolerance without the speed tax.

Public and Private Package Repositories

DC/OS makes it easy to install both community and proprietary packaged services.

The Mesosphere Universe Package Repository connects you with a library of open source industry-standard schedulers, services, and applications. Why reinvent the wheel if you don’t have to? Take advantage of community projects to handle batch job scheduling, highly available data storage, robust message queuing, and more.

DC/OS also supports installing from multiple package repositories: you can host your own private packages to be shared within your company or with your customers.

Cloud-Agnostic Installer

The DC/OS Installer makes it easy to install DC/OS on any cluster of physical or virtual machines.

For users with their own on-premise hardware or virtual machine provisioning infrastructure, the GUI or CLI Installer provides a quick, intuitive way to install DC/OS.

For users deploying to the public cloud, DC/OS offers several configurable cloud provisioning templates for AWS, Azure, and Packet.

For the advanced user, the Advanced Installer provides a scriptable, automatable interface to integrate with your prefered configuration management system.

Web and Command Line Interfaces

The DC/OS web and command line interfaces make it easy to monitor and manage the cluster and its services.

The DC/OS web interface lets you monitor resource allocation, running services, current tasks, component health, available packages, and more with intuitive browser-based navigation, real-time graphs, and interactive debugging tools.

The DC/OS command line interface provides control of DC/OS from the comfort of a terminal. It’s powerful, yet easily scriptable, with handy plugins to interact with installed services.

Elastic Scalability

DC/OS gives you the power to easily scale your services up and down with the turn of a dial.

Horizontal scaling is trivial in Marathon, as long as your service supports it. You can change the number of service instances at any time. DC/OS even lets you autoscale the number of instances based on session count, using the Marathon Load Balancer.

Vertical scaling is also supported in Marathon, allowing you to allocate more or less resources to services and automatically performing a rolling update to reschedule the instances without downtime.

Adding nodes to a DC/OS cluster is a snap too. The DC/OS Installer uses immutable artifacts that allow you to provision new nodes without having to recompile, reconfigure, or re-download component packages from flaky remote repositories.

High Availability

DC/OS is highly available and makes it easy for your services to be highly available too.

Mission-critical services require health monitoring, self-healing, and fault tolerance both for themselves and the platform and infrastructure they run on. DC/OS gives you multiple layers of protection.

To achieve self-healing, DC/OS services are monitored by Marathon and restarted when they fail. Even legacy services that don't support distribution or replication can be automatically restarted by Marathon to maximize uptime and reduce service interruption. On top of that, all core DC/OS components, including Marathon, are monitored by the DC/OS diagnostics service and restarted by `systemd` when they fail.

To achieve fault tolerance, DC/OS can run in multiple master configurations. This provides not just system-level fault tolerance but also scheduler-level fault tolerance. DC/OS can even survive node failure during an upgrade with no loss of service.

Zero Downtime Upgrades

DC/OS provides automation for updating services and the systems with zero downtime.

DC/OS services running on Marathon can all be updated with rolling, blue-green, or canary deployment patterns. If the update fails, roll it back with a single click. These powerful tools are critical for minimizing downtime and user interruption.

DC/OS itself also supports zero-downtime upgrades with its powerful installer. Stay up-to-date with the latest open source components with a single combined update.

Integration-Tested Components

DC/OS provides a well-tested set of open source components and bakes them all together with a single combined installer.

Mixing and matching open source components can be a pain. You never know which versions will work together or what the side effects of their interactions will be. Let the Mesos experts handle it for you! Get to production quickly and focus on the quality of your products, not the stability of your platform.

Service Discovery and Distributed Load Balancing

DC/OS includes several options for automating service discovery and load balancing.

Distributed services create distributed problems, but you don't have to solve them all yourself. DC/OS includes automatic DNS endpoint generation, an API for service lookup, transport layer (L4) virtual IP proxying for high speed internal communication, and application layer (L7) load balancing for external-facing services.

Control and Management plane for Distributed Load Balancers

Enterprise DC/OS provides a centralized management & control plane for service availability & performance monitoring.

While Distributed Load Balancers are ideal for service discovery and service availability of DC/OS Services, monitoring and managing them requires tooling and effort. Enterprise DC/OS comes with centralized control and management plane for DC/OS Distributed Load balancer that consists of an aggregation API which unifies all distributed engines into a single service centric view and single set of service health metrics. Enterprise DC/OS, also includes a Service Performance & health monitoring UI that helps monitor service performance as well as root cause service degradation issues and identify root causes.

Cluster Perimeter Security

DC/OS provides prescriptive design to ensure administrative and programmatic communication between DC/OS Clusters & any client (UI/Browsers, CLIs, API Clients etc.) happens over the Admin security zone and all requests are transported over SSL secured channel. DC/OS Master nodes are the entry point into the DC/OS Cluster within the Admin security zone and more specifically, the API Gateway is a component named 'Admin Router' that serves as reverse proxy managing all administrative connectivity into DC/OS Clusters.

Identity and Access Management Service

Enterprise DC/OS includes a built-in Identity and Access Management Service that allows our users to create Users and Groups and assign varying level of Authorization privileges to each user and group. Enterprise DC/OS supports following types of Users and Groups:

- Local Users
- Local Groups

- Remote LDAP Users
- Remote LDAP Groups (only for importing into local group)
- Remote SAML User
- Service User Account

Enterprise DC/OS IAM Service also includes support for authorization controls that can be assigned to each of the above principals/users. As of DC/OS 1.8, users can be given specific set of permissions in the form ‘Subject’ can perform ‘Action’ on ‘Object’ where ‘Object’ can be an API endpoint to a particular DC/OS Service to a Marathon Application group and ‘Action’ enumerates the set of actions that are possible on the Object such as “Create, Read, Update or Delete”.

External Identity Provider with LDAP, SAML & oAuth2

Enterprise DC/OS integrates Identity Providers that support LDAP v3 Interface (including Microsoft Active Directory) and SAML based identity providers such that you can import users external to DC/OS from your existing User Directory and manage authorization on those users and user group within DC/OS.

Cluster security with encrypted communication

Enterprise DC/OS is designed to run securely on-premises and in the cloud. To ensure cluster security, Enterprise DC/OS supports encrypted communication between DC/OS Cluster Internal components. This is achieved by ensuring that DC/OS runs with a Certificate Authority that issues certificates for DC/OS Master Nodes and all Agent nodes have an installed CA.crt at bootstrap time. This mechanism ensures all communication between the various services within DC/OS cluster are over secure SSL channels.

Workload Isolation with Container level authorization

Enterprise DC/OS supports fine-grained workload isolation to enable multiple business groups within an organization to run containers & workloads within a shared cluster but still be guaranteed that there is security isolation in addition to performance isolation provided by Linux cGroups between the varying workloads. Workload security isolation is performed by a DC/OS Authorization module that runs on every agent node and is responsible for making authorization checks against DC/OS IAM Service to verify that the user/owner of the workload is authorized to perform the action they are trying to execute anywhere within the cluster including on the Agent node.

IP per Container with Extensible Virtual Networks (SDN)

DC/OS comes built-in with support Virtual Networks leveraging Container Network Interface(CNI) standard. By default, there is one Virtual Network named ‘dos’ is created and any container that attaches to a Virtual Network, receives its own dedicated IP. This allows

users to run workloads that are not friendly to dynamically assigned ports and would rather bind the existing ports that is in their existing app configuration. Now, with support for dedicated IP/Container, workloads are free to bind to any port as every container has access to the entire available port range.

Network Isolation of Virtual Network Subnets

DC/OS now supports the creation of multiple virtual networks at install time and associate non-overlapping subnets with each of the virtual networks. Further, DC/OS users can program Network Isolation rules across DC/OS Agent nodes to ensure that traffic across Virtual Network subnets is isolated.

Components

ENTERPRISE DC/OS Updated: September 22, 2016

DC/OS is comprised of many individual open source components that are precisely configured to work together.

You can log into any host in the DC/OS cluster and view the currently running services by inspecting the `/etc/systemd/system/dcos.target.wants/` directory.

You can view the DC/OS component details in the https://github.com/dcos/dcos/ repo in the packages directory.

Component	Description
Admin Router Agent	This component (`dcos-adminrouter-agent`) is a high performance web server and a reverse proxy server that lists all of the agent nodes in your cluster.
Admin Router Master	This component is a high performance web server and a reverse proxy server that lists all of the master nodes in your cluster.
Admin Router Reloader	This component `dcos-adminrouter-reload.service` restarts the Admin Router Nginx server so that it can pick up new DNS resolutions, for example `master.mesos` and `leader.mesos`.
Admin Router Reloader Timer	This component `dcos-adminrouter-reload.timer` sets the Admin Router Reloader interval at once per hour.

Admin Router Service	This component is an open-source Nginx configuration created by Mesosphere that provides central authentication and proxy to DC/OS services within the cluster. The Admin Router service `dcos-adminrouter.service` is the core internal load balancer for DC/OS. Admin Router is a customized Nginx that proxies all of the internal services on port `80`.
Certificate Authority	This component `dcos-ca.service` is the DC/OS Certificate Authority feature. For more information, see the documentation.
Cluster ID	The cluster-id service generates a universally unique identifier (UUID) for each cluster. We use this ID to track cluster health remotely (if enabled). This remote tracking allows our support team to better assist our customers.
Diagnostics	This component `dcos-3dt.service` is the diagnostics utility for DC/OS systemd components. This service runs on every host, tracking the internal state of the systemd unit. The service runs in two modes, with or without the `-pull` argument. If running on a master host, it executes `/opt/mesosphere/bin/3dt -pull` which queries Mesos-DNS for a list of known masters in the cluster, then queries a master (usually itself) `:5050/statesummary` and gets a list of agents. From this complete list of cluster hosts, it queries all 3DT health endpoints `:1050/system/health/v1/health`. This endpoint returns health state for the DC/OS systemd units on that host. The master 3DT processes, along with doing this aggregation also expose `/system/health/v1/` endpoints to feed this data by `unit` or `node` IP to the DC/OS user interface.
Diagnostics socket	This component `dcos-3dt.socket` is the DC/OS Distributed Diagnostics Tool master API and aggregation socket.
DNS Dispatcher	This component `dcos-spartan.service` is an RFC5625 Compliant DNS Forwarder. It's job is to dual-dispatch DNS to multiple upstream resolvers, and to route DNS to the upstreams or Mesos DNS, depending on some rules.
DNS Dispatcher Watchdog	This component (`dcos-spartan-watchdog.service`) ensures that the DNS Dispatcher is running and healthy. If the DNS Dispatcher is unhealthy, this watchdog service kills it.
DNS Dispatcher Watchdog Timer	This component `dcos-spartan-watchdog.timer` wakes up the DNS Dispatcher Watchdog every 5 minutes, to see if DC/OS needs to restart DNS Dispatcher.
Erlang Port Mapping Daemon	This component `dcos-epmd.service` supports the internal DC/OS layer 4 load balancer that is called Minuteman.
Exhibitor	This component `dcos-exhibitor.service` is the Exhibitor supervisor for Zookeeper. DC/OS uses Exhibitor, a project from Netflix, to manage and automate the deployment of ZooKeeper.
Generate resolv.conf	This component `dcos-gen-resolvconf.service` dynamically provisions `/etc/resolv.conf` so that each cluster host can use Mesos-DNS to resolve task names to the IP and port addresses.
Generate resolv.conf Timer	This component `dcos-gen-resolvconf.timer` periodically updates the systemd-resolved for Mesos DNS.

History Service	This component `dcos-history-service.service` enables the DC/OS UI to display cluster usage statistics and stores the dashboard graph data for the UI. This data is stored on disk for 24 hours. Along with storing this data, the history service also exposes a HTTP API for the DC/OS user interface to query. All DC/OS cluster stats which involve memory, CPU, and disk usage are driven by this service. Without access to the History service, the UI will restart the graph timeline each time that a user opens the Dashboard tab in the UI. With access to the History service, the UI will show historical data, up to 60s prior, in the graph timeline.
Identity and Access Management	This component `dcos-bouncer.service` is the DC/OS Identity and Access Management feature. For more information, see the documentation.
Job	This component `dcos-metronome.service/` powers the DC/OS Jobs feature. For more information, see the documentation.
Layer 4 Load Balancer	This component `dcos-minuteman.service`, also known as Minuteman, is the DC/OS Layer 4 Load Balancer that enables multi-tier microservices architectures. For more information, see the documentation.
Logrotate Mesos Master	This component `dcos-logrotate-master.service` automatically manages compression, removal, and mailing of log files for Mesos master processes. This ensures DC/OS services don't overload cluster hosts with too much log data on disk.
Logrotate Mesos Slave	This component `dcos-logrotate-agent.service/` automatically rotates Mesos agent log files for agent nodes.
Logrotate Timer	These components `dcos-logrotate-agent.timer` and `dcos-logrotate-master.timer` set the logrotate interval at 2 minutes.
Marathon	This component `dcos-marathon.service` is the DC/OS Marathon instance which starts and monitors DC/OS applications and services.
Mesos Agent	This component `dcos-mesos-slave.service` is the mesos-slave process for private agent nodes.
Mesos Agent Public	This component `dcos-mesos-slave-public.service` is the mesos-slave process for public agent nodes.
Mesos DNS	This component `dcos-mesos-dns.service` provides service discovery within the cluster. Mesos-DNS is the internal DNS service `dcos-mesos-dns.service` for the DC/OS cluster. Mesos-DNS provides the namespace `$service.mesos` to all cluster hosts. For example, you can login to your leading mesos master with `ssh leader.mesos`.
Mesos Master	This component `dcos-mesos-master.service` is the mesos-master process that orchestrates agent tasks.
Mesos Persistent Volume Discovery	This component `dcos-vol-discovery-pub-agent.service` connects to existing Mesos volume mounts on agent nodes during installation. For more information on Mesos Persistent Volumes, see the documentation.
Network Metrics Aggregator	This component `dcos-networking_api.service` provides the DC/OS Network Metrics Aggregation Service and API. For more information, see the documentation.
Package service	This component `dcos-cosmos.service` is the internal packaging API service. This service is accessed every time that you run `dcos package install` from the CLI. This API deploys DC/OS packages from the DC/OS Universe to your DC/OS cluster.

REX-Ray	This component (`dcos-rexray.service`) is the REX-Ray storage method for enabling external persistent volumes in Marathon.
System Package Manager API	This component `dcos-pkgpanda-api.service` provides an API for installing and uninstalling DC/OS components.
System Package Manager API socket	This component `dcos-pkgpanda-api.socket` is the System Package Manager API socket.
Secrets Service	This component `dcos-secrets.service` secures important values like private keys, credentials, and database passwords. For more information, see the documentation.
Signal	This component `dcos-signal.service` sends a periodic ping back to Mesosphere with high-level cluster information to help improve DC/OS, and provides advanced monitoring of cluster issues. Signal queries the diagnostics service `/system/health/v1/report` endpoint on the leading master and sends this data to SegmentIO for use in tracking metrics and customer support.
Signal Timer	This component `dcos-signal.timer` sets the Signal component interval at once per hour.
Vault	This component `dcos-vault.service` is the storage backend for Secrets component. It manages the secure and durable processing of secrets submitted to the Secrets component.
Virtual Network Service	This component `dcos-navstar.service` is a daemon that provides virtual networking and DNS services. It is the network overlay orchestrator. For more information, see the documentation.

Concepts

ENTERPRISE DC/OS Updated: October 7, 2016

This page contains terms and definitions for DC/OS.

Admin Router

The Admin Router runs on the DC/OS master servers to provide a proxy for the admin parts of the cluster.

Agent node

A Mesos agent node runs a discrete Mesos task on behalf of a framework. It is an agent instance registered with the Mesos master. The synonym of agent node is worker or slave node. See also private and public agent nodes.

Bootstrap node

The node where a custom DC/OS installation is run. For more information, see the system requirements.

Cloud template

The cloud templates are optimized to run DC/OS. The templates are JSON-formatted text files that describe the resources and properties.

Mesos Containerizer

The Mesos Containerizer provides lightweight containerization and resource isolation of executors using Linux-specific functionality such as cgroups and namespaces.

Docker Containerizer

The Docker Containerizer enables launching docker containers using DC/OS.

Datacenter operating system

A new class of operating system that spans all of the machines in a datacenter or cloud and organizes them to act as one big computer.

DC/OS

The abbreviated form of the Datacenter Operating System.

DC/OS Cluster

A group of Mesos master and agent nodes.

DC/OS Marathon

The native Marathon instance that is the “init system” for DC/OS. It starts and monitors DC/OS applications and services.

DC/OS service

DC/OS services are applications that are packaged and available from the public GitHub package repositories. Available DC/OS services include Mesos frameworks and other applications.

Executor

A framework running on top of Mesos consists of two components: a scheduler that registers with the master to be offered resources, and an executor process that is launched on agent nodes to run the framework’s tasks. For more information about framework schedulers and executors, see the App/Framework development guide.

Exhibitor for ZooKeeper

DC/OS uses ZooKeeper, a high-performance coordination service to manage the installed DC/OS services. Exhibitor automatically configures your ZooKeeper installation on the master nodes during your DC/OS installation.

Framework

A Mesos framework is the combination of a Mesos scheduler and an optional custom executor. A framework receives resource offers describing CPU, RAM, etc. from the leading Mesos master, and allocates them for discrete tasks that can be launched on Mesos agent nodes. Mesosphere-certified Mesos frameworks, called DC/OS services, are packaged and available from public GitHub package repositories. DC/OS services include Mesosphere-certified Mesos frameworks and other applications.

Master

A Mesos master aggregates resource offers from all agent nodes and provides them to registered frameworks. For more details about the Mesos master, read about Mesos Master Configuration.

Mesos-DNS

Mesos-DNS is a DC/OS component that provides service discovery within the cluster. Mesos-DNS allows applications and services that are running on Mesos to find each other by using the domain name system (DNS), similar to how services discover each other throughout the Internet.

Package repository

DC/OS services are applications that are packaged and available from the public DC/OS package repositories that are hosted on GitHub.

Offer

An offer represents available resources (e.g. cpu, disk, memory) which an agent reports to the master and the master offers to the registered frameworks in some order.

Private agent node

Private agent nodes run DC/OS apps and services through a non-routable network that is only accessible from the admin zone or through the edgerouter from the public zone. By default DC/OS launches apps on private agent nodes. DC/OS agent nodes can be designated as public or private during installation. For more information, see the Network Security documentation.

Public agent node

Public agent nodes run DC/OS apps and services in a publicly accessible network. DC/OS agent nodes can be designated as public or private during installation. For more information see:

- Network Security
- Creating a public agent node

Slave

The synonym of slave node is worker or agent node. A Mesos agent node runs a discrete Mesos task on behalf of a framework. It is an agent instance registered with the Mesos master.

State abstraction

Mesos provides an abstraction for accessing storage for schedulers for Java and C++ only. This is the preferred method to access ZooKeeper for DC/OS services.

Task

A unit of work scheduled by a Mesos framework and executed on a Mesos agent. In Hadoop terminology, this is a “job”. In MySQL terminology, this is a “query” or “statement”. A task may simply be a Bash command, a Python script, or a complex AI application.

Working directory

A Mesos master requires a directory on the local file system to write replica logs to.

ZooKeeper

DC/OS uses ZooKeeper, a high-performance coordination service to manage the installed DC/OS services. Exhibitor automatically configures your ZooKeeper installation on the master nodes during your DC/OS installation.

High Availability

ENTERPRISE DC/OS Updated: October 7, 2016

This document discusses the high availability features in DC/OS and best practices for building highly available applications on DC/OS.

Leader/follower architecture

A common pattern in highly available systems is the leader/follower concept. This is also sometimes referred to as: master/slave, primary/replica, or some combination thereof. Generally speaking, this architecture is used when you have one authoritative process, with N standby processes. In some systems, the standby processes might also be capable of serving requests or performing other operations. For example, when running a database like MySQL with a master and replica, the replica is able to serve read-only requests, but it cannot accept writes (only the master will accept writes).

In DC/OS, a number of components follow the leader/follower pattern. We'll discuss some of them here and how they work.

Mesos

Mesos may be run in high availability mode, which requires running 3 or 5 masters. When run in HA mode, one master will become elected as the leader, while the other masters will become followers. Each master has a replicated log which contains some state about the

cluster. The leading master is elected by using ZooKeeper to perform leader election. For more detail on this, see the Mesos HA documentation.

Marathon

Marathon may be operated in HA mode, which allows running multiple Marathon instances (at least 2 for HA), with one elected leader. Marathon will use ZooKeeper for leader election, and the followers will not accept writes or API requests, but will proxy all API requests to the leading Marathon instance.

ZooKeeper

ZooKeeper is used by numerous services in DC/OS to provide consistency. ZooKeeper can be used as a distributed locking service, a state store, and a messaging system. ZooKeeper uses Paxos-like log replication and a leader/follower architecture to maintain consistency across multiple ZooKeeper instances. For a more detailed explanation of how ZooKeeper works, check out the ZooKeeper internals document.

Fault domain isolation

Fault domain isolation is an important part of building HA systems. In order to correctly handle failure scenarios, systems must be distributed across fault domains in order survive outages. There are different types of fault domains, a few examples of which are:

- Physical domains: this includes machine, rack, datacenter, region, availability zone, and so on.
- Network domains: machines within the same network may be subject to network partitions. For example, a shared network switch may fail or have invalid configuration.

With DC/OS, it's recommended that masters be distributed across racks for HA, but not across DCs or regions. Agents may be distributed across regions/DCs, and it's recommended that you tag agents with attributes to describe their location. Synchronous services like ZooKeeper should also remain within the same region/DC to reduce network latency.

For applications which require HA, they should also be distributed across fault domains. With Marathon, this can be accomplished by using the `UNIQUE` and `GROUP_BY` constraints operator.

Separation of concerns

HA services should be decoupled, with responsibilities divided amongst services. For example, web services should be decoupled from databases and shared caches.

Eliminating single points of failure

Single points of failure come in many forms. A service like ZooKeeper, for example, can become a single point of failure when every service in your system shares one ZooKeeper cluster. You can reduce risks by running multiple ZooKeeper clusters for separate services. With DC/OS, there's an Exhibitor package included which makes this easy:

```
$ dcos package install exhibitor
```

Other common single points of failure include: single database instances (like a MySQL), one-off services, and non-HA load balancers.

Fast failure detection

Fast failure detection comes in many forms. Services like ZooKeeper can be used to provide failure detection, such as detecting network partitions or host failures. Service health checks can also be used to detect certain types of failures. As a matter of best practice, services *should* expose health check endpoints, which can be used by services like Marathon.

Fast failover

When failures do occur, failover should be as fast as possible. Fast failover can be achieved by:

- Using an HA load balancer like Marathon-LB, or Minuteman for internal layer 4 load balancing.
- Building apps in accordance with the 12-factor app manifesto.
- Following REST best-practices when building services: in particular, avoiding storing client state on the server between requests.

A number of DC/OS services follow the fail-fast pattern in the event of errors. Specifically, both Mesos and Marathon will shut down in the case of unrecoverable conditions such as losing leadership.

Telemetry

ENTERPRISE DC/OS Updated: September 9, 2016

To continuously improve the DC/OS experience, a telemetry component is included that reports anonymous usage data to Mesosphere. This data is used to monitor the reliability of core DC/OS components, installations, user interface, and to find out which features are most popular.

- Core telemetry
- User interface telemetry

Core telemetry

The DC/OS Signal component queries the diagnostics service `/system/health/v1/report` endpoint on the leading master and sends this data to Segment which Mesosphere then uses to track usage metrics and customer support.

The information collected by the Signal component is separated into these categories: Diagnostics, Mesos, and Package service.

For each category this data is collected:

Type	Description
anonymousId	This is an anonymous ID that is created for every cluster at startup. This ID persists across your cluster. For example: `"anonymousId": "70b28f00-e38f-41b2-a723-aab344f535b9`
clusterId	This is the `anonymousID` value that is created for every cluster at startup. This ID persists across your cluster. For example: `"clusterId": "70b28f00-e38f-41b2-a723-aab344f535b9"`
customerKey (Enterprise DC/OS)	This is the Enterprise DC/OS customer key. Customer keys are delivered via email to the Authorized Support Contact. For example: `"customerKey": "ab1c23de-45f6-7g8h-9012-i345j6k7lm8n",`
event	This is the category that appears in Segment. Possible values are `package_list` (Package service), `health` (Diagnostics), and `mesos_track` (Mesos). For example: `"event": "package_list"`
environmentVersion	This is the version of DC/OS. For example, if you are using DC/OS 1.8: `"environmentVersion": "1.8",`
provider	This is the platform that DC/OS is running on. Possible values are `aws`, `on-prem`, and `azure`. For example, if you are running on AWS: `"provider": "aws",`
source	This is a hard-coded setting that indicates a cluster. For example: `"source": "cluster",`
variant	This indicates whether the cluster is DC/OS or Enterprise DC/OS. For example, if you are using DC/OS: `"variant": "open"`

Diagnostics

This information is collected from the DC/OS Diagnostics component. For every systemd

unit, the following information is collected, where `<UNIT_NAME>` is component name:

```
"health-unit-dcos-<UNIT_NAME>-total": 3, "health-unit-dcos-<UNIT_NAME>-unhealthy": 0,
```

Mesos

This information is collected from the DC/OS Mesos Master component.

Type	Description
agents_active	Number of active agents. For example: `"agents_active": 2,`
agents_connected	Number of connected agents. For example: `"agents_connected": 2,`
cpu_total	Number of CPUs available. For example: `"cpu_total": 8,`
cpu_used	Number of allocated CPUs. For example: `"cpu_used": 0,`
disk_total	Disk space available in MB. For example: `"disk_total": 71154,`
disk_used	Allocated disk space in MB. For example: `"disk_used": 0,`
framework_count	Number of installed DC/OS services. For example: `"framework_count": 2,`
frameworks	Which DC/OS services are installed. For example: `"frameworks": [ { "name": "marathon" }, { "name": "metronome" } ],`
mem_total	Memory available in MB. For example: `"mem_total": 28036,`
mem_used	Memory allocated in MB. For example: `"mem_used": 0,`
task_count	Number of tasks. For example: `"task_count": 0,`

Package service

This information is collected from the DC/OS Package service component.

Type	Description

Which packages are installed. For example, if you had Kafka and Spark:

```
"package_list": [
{
"name": "kafka"
```

package_list},

```
{
"name": "spark"
}
],
```

Here is an example of the JSON telemetry report that is collected:


```
{
    "cosmos": {
        "properties": {
            "clusterId": "70b28f00-e38f-41b2-a723-aab344f535b9",
            "customerKey": "",
            "environmentVersion": "1.8",
            “package_list”: [
            {
            “name”: “kafka”
            },
            {
            “name”: “spark”
            }
            ],
            "provider": "aws",
            "source": "cluster",
            "variant": "open"
        },
        "anonymousId": "70b28f00-e38f-41b2-a723-aab344f535b9",
        "event": "package_list"
    },
    "diagnostics": {
        "properties": {
            "clusterId": "70b28f00-e38f-41b2-a723-aab344f535b9",
            "customerKey": "",
            "environmentVersion": "1.8",
            "health-unit-dcos-3dt-service-total": 3,
            "health-unit-dcos-3dt-service-unhealthy": 0,
            "health-unit-dcos-3dt-socket-total": 2,
            "health-unit-dcos-3dt-socket-unhealthy": 0,
            "health-unit-dcos-adminrouter-agent-service-total": 2,
            "health-unit-dcos-adminrouter-agent-service-unhealthy": 0,
            "health-unit-dcos-adminrouter-reload-service-total": 3,
            "health-unit-dcos-adminrouter-reload-service-unhealthy": 0,
            "health-unit-dcos-adminrouter-reload-timer-total": 3,
            "health-unit-dcos-adminrouter-reload-timer-unhealthy": 0,
            "health-unit-dcos-adminrouter-service-total": 1,
            "health-unit-dcos-adminrouter-service-unhealthy": 0,
            "health-unit-dcos-cosmos-service-total": 1,
            "health-unit-dcos-cosmos-service-unhealthy": 0,
            "health-unit-dcos-epmd-service-total": 3,
            "health-unit-dcos-epmd-service-unhealthy": 0,
            "health-unit-dcos-exhibitor-service-total": 1,
            "health-unit-dcos-exhibitor-service-unhealthy": 0,
            "health-unit-dcos-gen-resolvconf-service-total": 3,
            "health-unit-dcos-gen-resolvconf-service-unhealthy": 0,
            "health-unit-dcos-gen-resolvconf-timer-total": 3,
```

```
        "health-unit-dcos-gen-resolvconf-timer-unhealthy": 0,
        "health-unit-dcos-history-service-total": 1,
        "health-unit-dcos-history-service-unhealthy": 0,
        "health-unit-dcos-logrotate-agent-service-total": 2,
        "health-unit-dcos-logrotate-agent-service-unhealthy": 0,
        "health-unit-dcos-logrotate-agent-timer-total": 2,
        "health-unit-dcos-logrotate-agent-timer-unhealthy": 0,
        "health-unit-dcos-logrotate-master-service-total": 1,
        "health-unit-dcos-logrotate-master-service-unhealthy": 0,
        "health-unit-dcos-logrotate-master-timer-total": 1,
        "health-unit-dcos-logrotate-master-timer-unhealthy": 0,
        "health-unit-dcos-marathon-service-total": 1,
        "health-unit-dcos-marathon-service-unhealthy": 0,
        "health-unit-dcos-mesos-dns-service-total": 1,
        "health-unit-dcos-mesos-dns-service-unhealthy": 0,
        "health-unit-dcos-mesos-master-service-total": 1,
        "health-unit-dcos-mesos-master-service-unhealthy": 0,
        "health-unit-dcos-mesos-slave-public-service-total": 1,
        "health-unit-dcos-mesos-slave-public-service-unhealthy": 0,
        "health-unit-dcos-mesos-slave-service-total": 1,
        "health-unit-dcos-mesos-slave-service-unhealthy": 0,
        "health-unit-dcos-metronome-service-total": 1,
        "health-unit-dcos-metronome-service-unhealthy": 0,
        "health-unit-dcos-minuteman-service-total": 3,
        "health-unit-dcos-minuteman-service-unhealthy": 0,
        "health-unit-dcos-navstar-service-total": 3,
        "health-unit-dcos-navstar-service-unhealthy": 0,
        "health-unit-dcos-oauth-service-total": 1,
        "health-unit-dcos-oauth-service-unhealthy": 0,
        "health-unit-dcos-pkgpanda-api-service-total": 3,
        "health-unit-dcos-pkgpanda-api-service-unhealthy": 0,
        "health-unit-dcos-pkgpanda-api-socket-total": 3,
        "health-unit-dcos-pkgpanda-api-socket-unhealthy": 0,
        "health-unit-dcos-rexray-service-total": 2,
        "health-unit-dcos-rexray-service-unhealthy": 0,
        "health-unit-dcos-signal-service-total": 1,
        "health-unit-dcos-signal-service-unhealthy": 0,
        "health-unit-dcos-signal-timer-total": 3,
        "health-unit-dcos-signal-timer-unhealthy": 0,
        "health-unit-dcos-spartan-service-total": 3,
        "health-unit-dcos-spartan-service-unhealthy": 0,
        "health-unit-dcos-spartan-watchdog-service-total": 3,
        "health-unit-dcos-spartan-watchdog-service-unhealthy": 0,
        "health-unit-dcos-spartan-watchdog-timer-total": 3,
        "health-unit-dcos-spartan-watchdog-timer-unhealthy": 0,
        "health-unit-dcos-vol-discovery-priv-agent-service-total": 1,
        "health-unit-dcos-vol-discovery-priv-agent-service-unhealthy": 0,
        "health-unit-dcos-vol-discovery-pub-agent-service-total": 1,
        "health-unit-dcos-vol-discovery-pub-agent-service-unhealthy": 0,
        "provider": "aws",
        "source": "cluster",
        "variant": "open"
    },
    "anonymousId": "70b28f00-e38f-41b2-a723-aab344f535b9",
    "event": "health"
},
"mesos": {
    "properties": {
        "agents_active": 2,
        "agents_connected": 2,
        "clusterId": "70b28f00-e38f-41b2-a723-aab344f535b9",
        "cpu_total": 8,
```

```
            "cpu_used": 0,
            "customerKey": "",
            "disk_total": 71154,
            "disk_used": 0,
            "environmentVersion": "1.8",
            "framework_count": 2,
            "frameworks": [
                {
                    "name": "marathon"
                },
                {
                    "name": "metronome"
                }
            ],
            "mem_total": 28036,
            "mem_used": 0,
            "provider": "aws",
            "source": "cluster",
            "task_count": 0,
            "variant": "open"
        },
        "anonymousId": "70b28f00-e38f-41b2-a723-aab344f535b9",
        "event": "mesos_track"
    }
}
```


User interface telemetry

The DC/OS UI sends two types of notifications to Segment which Mesosphere then uses to track usage metrics and customer support:

- Login information
- The pages you've viewed while navigating the UI

Service Discovery

ENTERPRISE DC/OS Updated: October 7, 2016

There are two levels of service discovery in DC/OS. Along with every task that runs on DC/OS being provided a well known DNS name, anyone can request a well known VIP that

enables clients to have a single configuration value.

VIPs

You can assign a VIP or VIPs to one of your services by following the steps in the Service Discovery section.

Mesos-DNS

Every task started by DC/OS gets a well known DNS name. You can even enumerate every DNS name in your cluster. For a Marathon service named “testing”, you can find where it is running via:

```
dig testing.marathon.mesos
```

Take a look at the mesos-DNS documentation for a more in-depth look at how Mesos-DNS is working and what it is doing for you.

Roadmap

ENTERPRISE DC/OS Updated: October 7, 2016

This roadmap provides a high level overview of the themes that the DC/OS project is focusing on in the near, medium and long term.
Please see the design docs and JIRAs for additional details on each item.

Near Term

Pods

Pods enable the popular “sidecar” pattern where multiple containers are scheduled together

on the same host and with shared resources. Pods allows co-scheduling of monitoring agents, service registration clients, and more with the main application process.

Metrics API

The Metrics API provides a way for any component of DC/OS to publish its own metrics. Metrics can be forwarded to standard metrics aggregation tools such as Graphite, Grafana, InfluxDB, and Prometheus.

Unified Logging

Aggregate logs from all parts of DC/OS via Journald. This includes master, agent, and service logs. Plugins and tutorials for integrating with log aggregation systems like Splunk or ELK.

System Services

Run a service on every node in the cluster to support add-ons that need a daemon process to be present cluster-wide. Examples are monitoring agents, log ingestion, security monitoring and enforcement, networking.

Package Profiles

Multiple configuration profiles in Universe packages, to support dev/prod configurations and small/medium/large setups.

Package Dependencies

Support for dependencies in Universe packages.

GPU Support

Support for discovery, isolation, and consumption of GPUs. Users will be able to request machines with GPU support for their tasks. DC/OS provides isolation so that multiple containers cannot interfere with each other.

DC/OS on Windows

Support running DC/OS nodes on Windows machines, and support for scheduling workloads in Windows containers.

Long Term

Debugging

Tools for debugging applications at runtime, and integrating with existing debugging tools such as gdb and IDEs.

Autoscaling

Automatic scale-out of container-based applications based on configurable metrics such as response time and throughput.

Feature Maturity

ENTERPRISE DC/OS Updated: October 7, 2016

The purpose of the feature maturity phases is to educate customers, partners, and Mesosphere field and support organizations about the maturity and quality of features.

Criteria

Completeness

Functionality: Completeness of the feature implementation.

Interfaces: Feature has an API with deprecation cycle, CLI, and UI.

Documentation: Feature has appropriate documentation. e.g., Admin Guide, Developer Guide, Release Notes.

Quality

Functional Test: Feature is validated for correctness.

System Test: Feature is validated to meet scalability, reliability, and performance requirements through a combination of load, soak, stress, spike, fault, and configuration tests.

Mesosphere Dogfooding: Feature in-use in Mesosphere production environment.

Phases

Experimental

Use these features at your own risk. We might add, change, or delete any functionality.

Completeness

- Feature may be incomplete
- API may be incomplete and is subject to change without warning or deprecation cycle
- User interfaces may be missing or incomplete
- Documentation may be missing or incomplete

Quality

- Limited or no functional test
- Limited or no system test
- Limited or no Mesosphere dogfooding

Preview

We might add, change, or delete any functionality.

Completeness

- Feature is complete
- API may be incomplete and changes may not be subject to deprecation cycle
- User interfaces may be missing or incomplete
- Documentation may be incomplete

Quality

- Robust functional test
- Limited or no system test
- Limited or no Mesosphere dogfooding

Stable

Completeness

- Feature is complete

- API is complete and changes are subject to deprecation cycle
- User Interfaces are complete
- Documentation is complete

Quality

- Robust functional test
- Robust performance testing
- Robust fault testing
- Robust Mesosphere dogfooding