Desmistificando Microsserviços e DevOps: Projetando Arquiteturas Efetivamente Escaláveis

Prof. Vinicius Cardoso Garcia
vcg@cin.ufpe.br :: @vinicius3w :: assertlab.com

[IF1004] - Seminários em SI 3
https://github.com/vinicius3w/if1004-DevOps

Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-Compartilhalgual 3.0 Não Adaptada

Mais informações visite

http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt

Managing Dockerized Microservices with Mesos and Marathon

Context

- In an Internet-scale microservices deployment, it is not easy to manage thousands of dockerized microservices
- It is essential to have an infrastructure abstraction layer and a strong cluster control platform to successfully manage Internet-scale microservice deployments
- This lecture will explain the need and use of Mesos and Marathon as an infrastructure abstraction layer and a cluster control system, respectively, to achieve optimized resource usage in a cloud-like environment when deploying microservices at scale

The missing pieces

- Docker helped package the JVM runtime and OS parameters along with the application
- There is no special consideration required when moving dockerized microservices from one environment to another
- The REST APIs provided by Docker have simplified the life cycle manager's interaction with the target machine in starting and stopping artifacts

The missing pieces

- In a large-scale deployment, with hundreds and thousands of Docker containers, we need to ensure that Docker containers run with their own resource constraints, such as memory, CPU, and so on
- There may be rules set for Docker deployments, such as replicated copies of the container should not be run on the same machine
- A mechanism needs to be in place to optimally use the server infrastructure to avoid incurring extra cost

The missing pieces

- There are organizations that deal with billions of containers, managing them manually is next to impossible
- In the context of large-scale Docker deployments, some of the key questions to be answered are
    1. How do we manage thousands of containers?
    2. How do we monitor them?
    3. How do we apply rules and constraints when deploying artifacts?
    4. How do we ensure that we utilize containers properly to gain resource efficiency?
    5. How do we ensure that at least a certain number of minimal instances are running at any point in time?
    6. How do we ensure dependent services are up and running?
    7. How do we do rolling upgrades and graceful migrations?
    8. How do we roll back faulty deployments?

The missing pieces

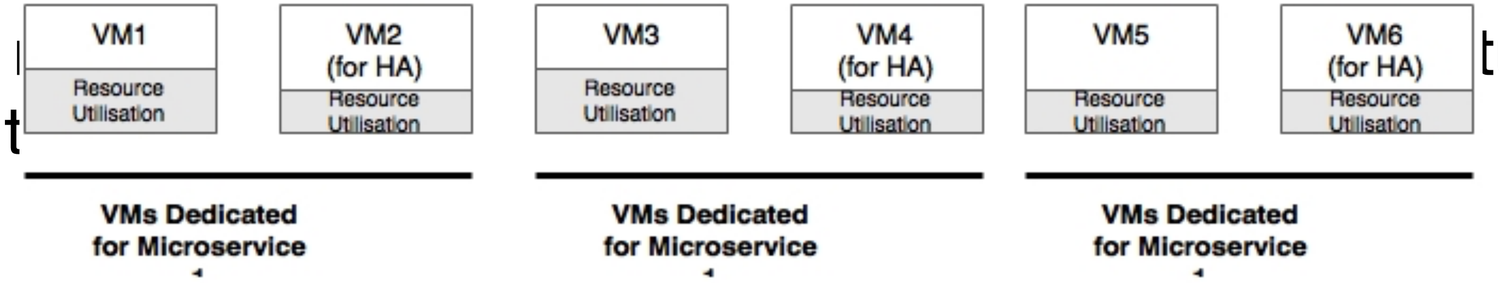
- All these questions point to the need to have a solution to address two key capabilities
  I. A cluster abstraction layer that provides a uniform abstraction over many physical or virtual machines
  II. A cluster control and init system to manage deployments intelligently on top of the cluster abstraction

The missing pieces

- The life cycle manager is ideally placed to deal with these situations
- One can add enough intelligence to the life cycle manager to solve these issues
- However, before attempting to modify the life cycle manager, it is important to understand the role of cluster management solutions a bit more.

Why cluster management is important

- As microservices break applications into different micro-applications, many developers request more server nodes for deployment
- In order to manage microservices properly, developers tend to deploy one microservice per VM, which further drives down the resource utilization
- In many cases, this results in an overallocation of CPUs and memory
- In many deployments, the high-availability requirements of microservices force engineers to add more and more service instances for redundancy
- In general, I ture compared t



Why cluster management is important

- In order to address the issue stated before, we need a tool that is capable of
  - Automating a number of activities, such as the allocation of containers to the infrastructure efficiently and keeping it transparent to developers and administrators
  - Providing a layer of abstraction for the developers so that they can deploy their application against a data center without knowing which machine is to be used to host their applications
  - Setting rules or constraints against deployment artifacts
  - Offering higher levels of agility with minimal management overheads for developers and administrators, perhaps with minimal human interaction
  - Building, deploying, and managing the application's cost effectively by driving a maximum utilization of the available resources

What does cluster management do?

- Typical cluster management tools help virtualize a set of machines and manage them as a single cluster
- Cluster management tools also help move the workload or containers across machines while being transparent to the consumer
- The fundamental function of these cluster management tools is to abstract the actual server instance from the application developers and administrators
  - Help the self-service and provisioning of infrastructure rather than requesting the infrastructure teams to allocate the required machines with a predefined specification
  - Machines are no longer provisioned upfront and preallocated to the applications

Key capabilities of cluster management software

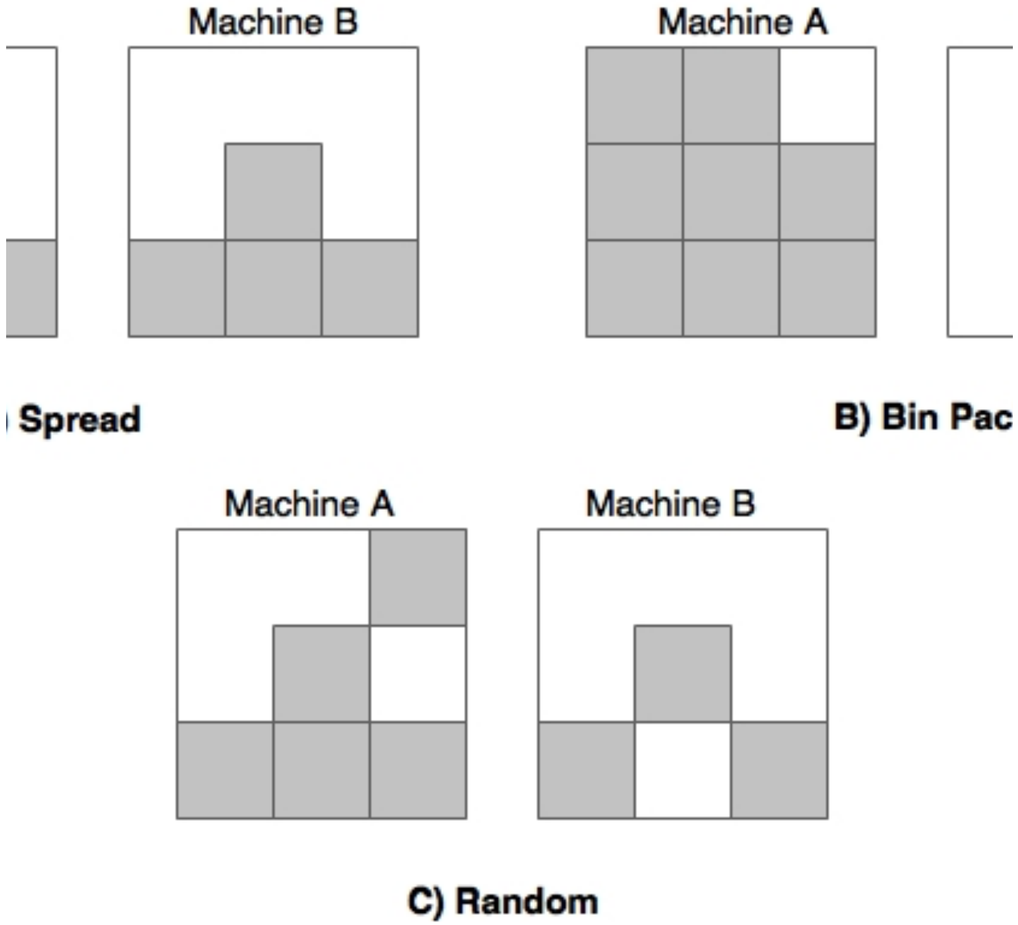
- **Cluster** management: It manages a cluster of VMs and physical machines as a single large machine
- **Deployments**: It handles the automatic deployment of applications and containers with a large set of machines
- **Scalability**: It handles the automatic and manual scalability of application instances as and when required, with optimized utilization as the primary goal
- **Health**: It manages the health of the cluster, nodes, and applications
- **Infrastructure abstraction**: It abstracts the developers from the actual machine on which the applications are deployed

Key capabilities of cluster management software

- **Resource optimization**: The inherent behavior of these tools is to allocate container workloads across a set of available machines in an efficient way
- **Resource allocation**: It allocates servers based on resource availability and the constraints set by application developers
- **Service availability**: It ensures that the services are up and running somewhere in the cluster
- **Agility**: These tools are capable of quickly allocating workloads to the available resources or moving the workload across machines if there is change in resource requirements
- **Isolation**: Some of these tools provide resource isolation out of the box. Hence, even if the application is not containerized, resource isolation can be still achieved

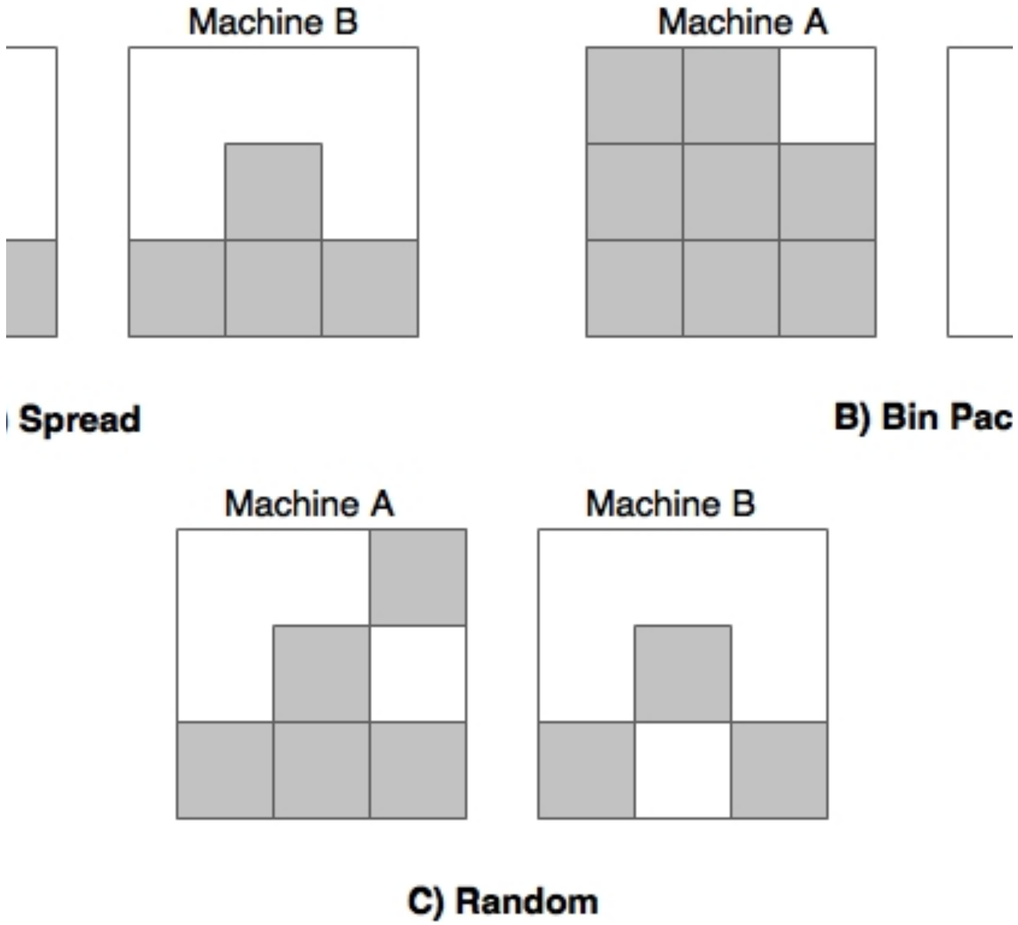
Key capabilities of cluster management software

- A variety of algorithms are used for resource allocation, ranging from simple algorithms to complex algorithms, with machine learning and artificial intelligence
- The common algorithms used are random, bin packing, and spread
- Constraints set against applications will override the default algorithms based on resource availability



Key capabilities of cluster management software

- **Spread**: This algorithm performs the allocation of workload equally across the available machines
- **Bin packing**: This algorithm tries to fill in data machine by machine and ensures the maximum utilization of machines. Bin packing is especially good when using cloud services in a pay-as-you-use style
- **Random**: This algorithm randomly chooses machines and deploys containers on randomly selected machines



Relationship with microservices

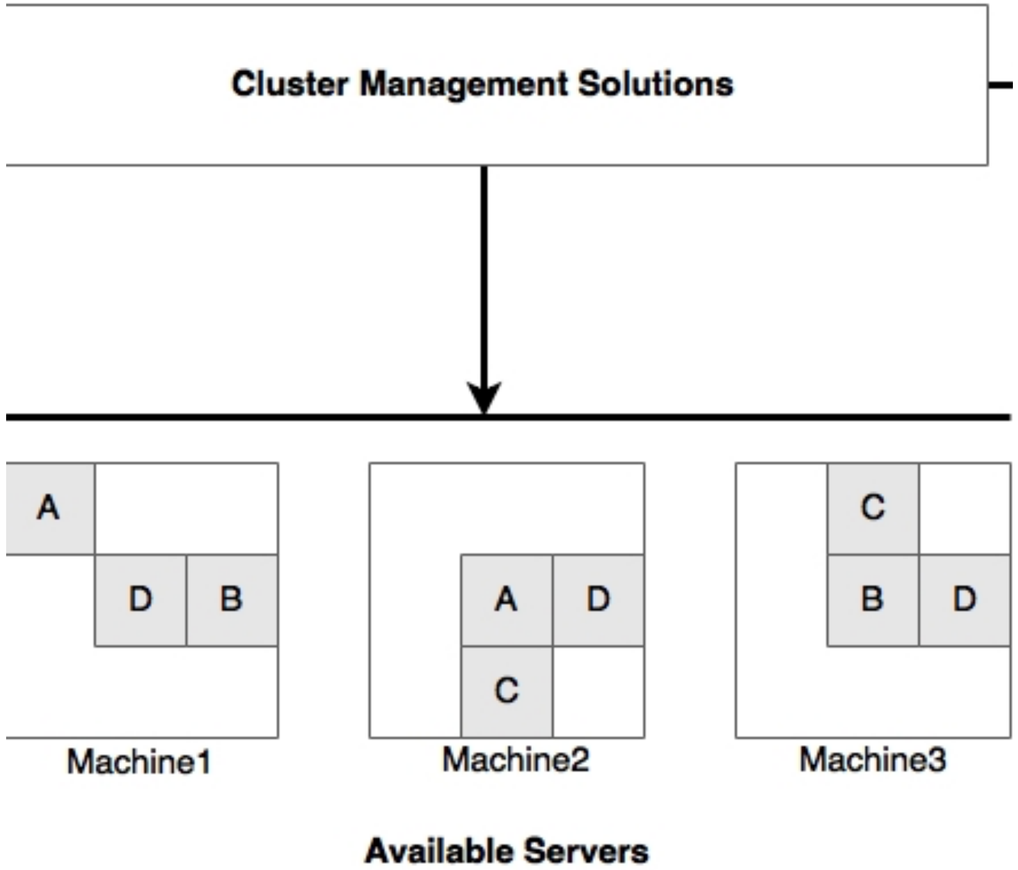
- The infrastructure of microservices, if not properly provisioned, can easily result in oversized infrastructures and, essentially, a higher cost of ownership
- As Spring Cloud-based microservices are location unaware, these services can be deployed anywhere in the cluster
  - Whenever services come up, they automatically register to the service registry and advertise their availability
- This way, the application supports a full fluid structure without preassuming a deployment topology

Relationship with virtualization

- Cluster management solutions are different from server virtualization solutions in many aspects
- Cluster management solutions run on top of VMs or physical machines as an application component

Cluster management solutions

- There are many cluster management software tools available
- It is unfair to do an apple-to-apple comparison between them
- Even though there are no one-to-one components, there are many areas of overlap in capabilities between them
- In many situations, organizations use a combination of one or more of these tools to fulfill their requirements



Docker Swarm

- Docker's native cluster management solution
- Swarm provides a native and deeper integration with Docker and exposes APIs that are compatible with Docker's remote APIs
- Docker Swarm logically groups a pool of Docker hosts and manages them as a single large Docker virtual host
- Instead of application administrators and developers deciding on which host the container is to be deployed in, this decision making will be delegated to Docker Swarm
- Docker Swarm will decide which host to be used based on the bin packing and spread algorithms
- Docker Swarm works with the concepts of **manager** and **nodes**. A **manager** is the single point for administrations to interact and schedule the Docker containers for execution. **Nodes** are where Docker containers are deployed and run.

Kubernetes (k8s)

- Comes from Google's engineering, is written in the Go language, and is battle-tested for large-scale deployments at Google
- Similar to Swarm, Kubernetes helps manage containerized applications across a cluster of nodes
- Kubernetes helps automate container deployments, scheduling, and the scalability of containers
- The Kubernetes architecture has the concepts of **master**, **nodes**, and **pods**
  - The **master** and **nodes** together form a Kubernetes cluster
  - The **master** node is responsible for allocating and managing workload across a number of nodes
  - **Nodes** are nothing but a VM or a physical machine
  - **Nodes** are further subsegmented as **pods**
  - A **node** can host multiple **pods**
  - One or more containers are grouped and executed inside a **pod**

Apache Mesos

- Mesos is an open source framework originally developed by the University of California at Berkeley and is used by Twitter at scale primarily to manage the large Hadoop ecosystem
- Is more of a resource manager that relays on other frameworks to manage workload execution
- Sits between the operating system and the application, providing a logical cluster of machines
- Is a distributed system kernel that logically groups and virtualizes many computers to a single large machine
- Has the concepts of the **master** and **slave** nodes. Similar to the earlier solutions, **master** nodes are responsible for managing the cluster, whereas **slaves** run the workload
  - Internally uses ZooKeeper for cluster coordination and storage
  - Supports the concept of frameworks that are responsible for scheduling and running noncontainerized applications and containers
  - Marathon, Chronos, and Aurora are popular frameworks for the scheduling and execution of applications
  - Netflix Fenzo is another open source Mesos framework. Interestingly, Kubernetes also can be used as a Mesos framework.

Nomad

- Nomad is a cluster management system that abstracts lower-level machine details and their locations
- Nomad has a simpler architecture compared to the other solutions explored earlier
- Nomad has the concept of **servers**, in which all **jobs** are managed
- One **server** acts as the **leader**, and others act as **followers**
- Nomad has the concept of **tasks**, which is the smallest unit of work
  - **Tasks** are grouped into **task groups**
  - A **task group** has **tasks** that are to be executed in the same location
  - One or more **task groups** or **tasks** are managed as **jobs**
- Nomad supports many workloads, including Docker, out of the box
- Nomad also supports deployments across data centers and is region and data center aware

Fleet

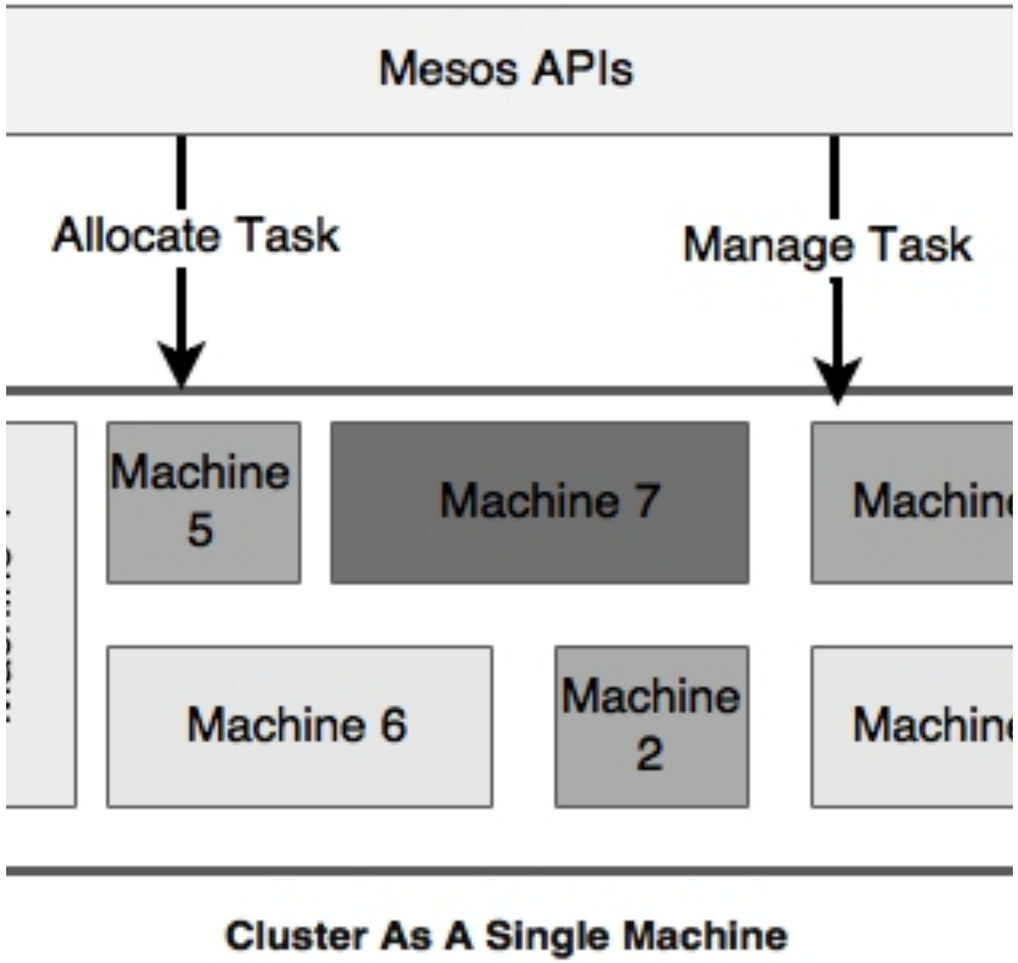
- Fleet is a cluster management system from CoreOS
- It runs on a lower level and works on top of systemd
- Fleet can manage application dependencies and make sure that all the required services are running somewhere in the cluster
- If a service fails, it restarts the service on another host. Affinity and constraint rules are possible to supply when allocating resources
- Fleet has the concepts of **engine** and **agents**
  - There is only one **engine** at any point in the cluster with multiple **agents**
  - Tasks are submitted to the engine and agent run these tasks on a cluster machine

Cluster management with Mesos and Marathon

- Many organizations choose Kubernetes or Mesos with a framework such as Marathon
  - In most cases, Docker is used as a default containerization method to package and deploy workloads
- For the rest of this lecture, we will show how Mesos works with Marathon to provide the required cluster management capability
- Mesos is used by many organizations, including Twitter, Airbnb, Apple, eBay, Netflix, PayPal, Uber, Yelp, and many others

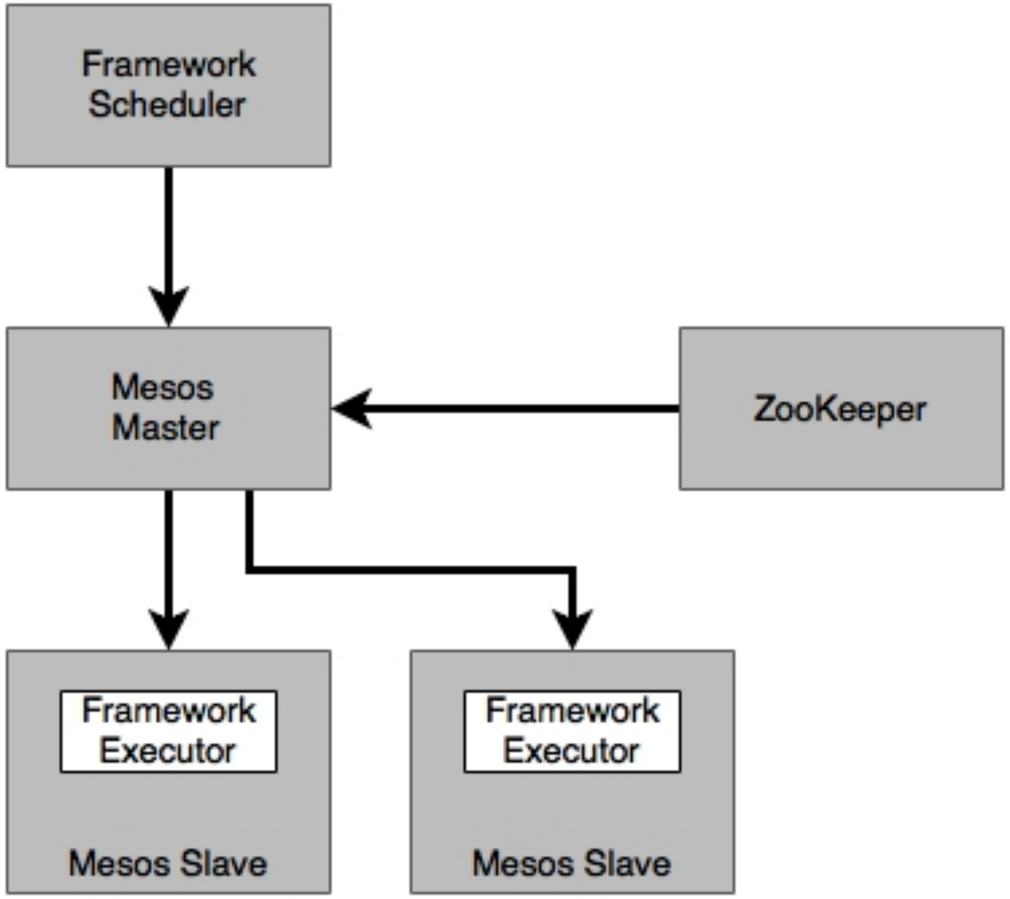
Diving deep into Mesos

- Mesos can be treated as a data center kernel
- In order to run multiple tasks on one node, Mesos uses resource isolation concepts
- Mesos relies on the Linux kernel's **cgroups** to achieve resource isolation similar to the container approach
  - It also supports containerized isolation using Docker
- Mesos supports both batch workload as well as the OLTP kind of workloads



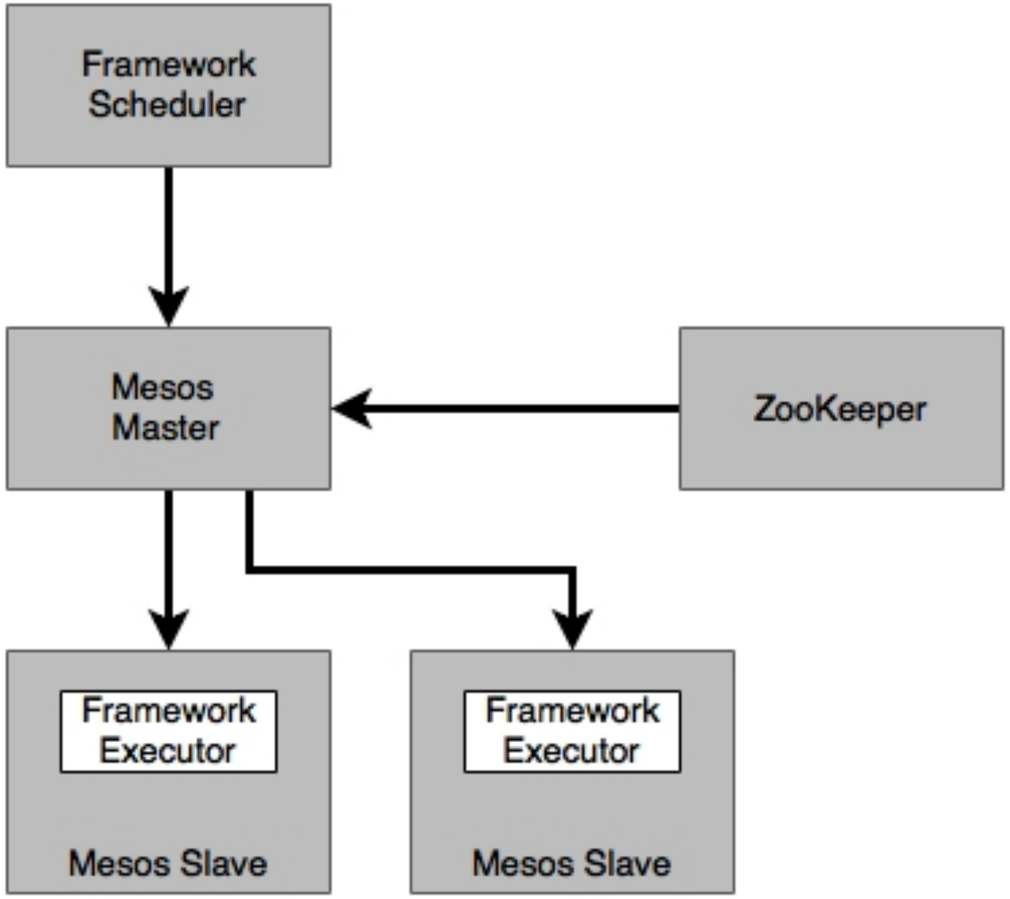
The Mesos architecture

- **Master**: The Mesos master is responsible for managing all the Mesos slaves
  - gets information on the resource availability from all slave nodes and take the responsibility of filling the resources appropriately based on certain resource policies and constraints
  - preempts available resources from all slave machines and pools them as a single large machine



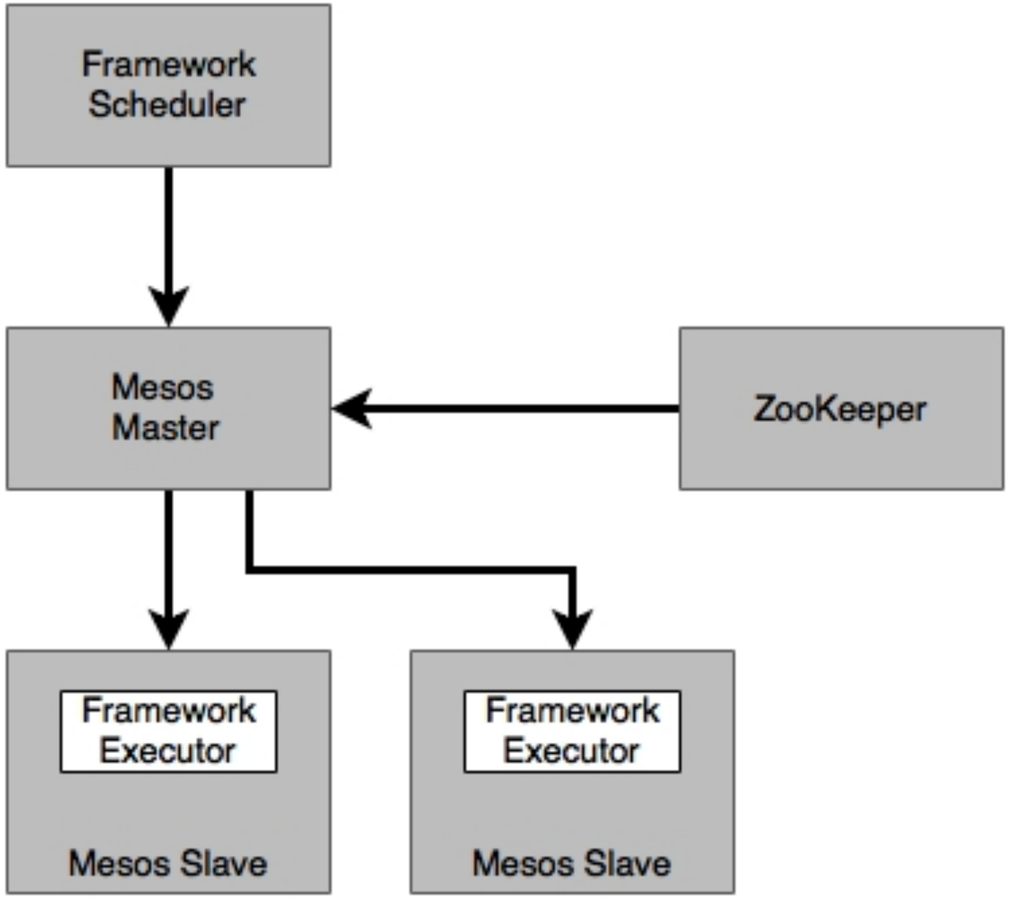
The Mesos architecture

- **Slave**: Mesos slaves are responsible for hosting task execution frameworks
  - Tasks are executed on the slave nodes
  - Mesos slaves can be started with attributes as key-value pairs, such as data center = X
  - This is used for constraint evaluations when deploying workloads
  - Slave machines share resource availability with the Mesos master



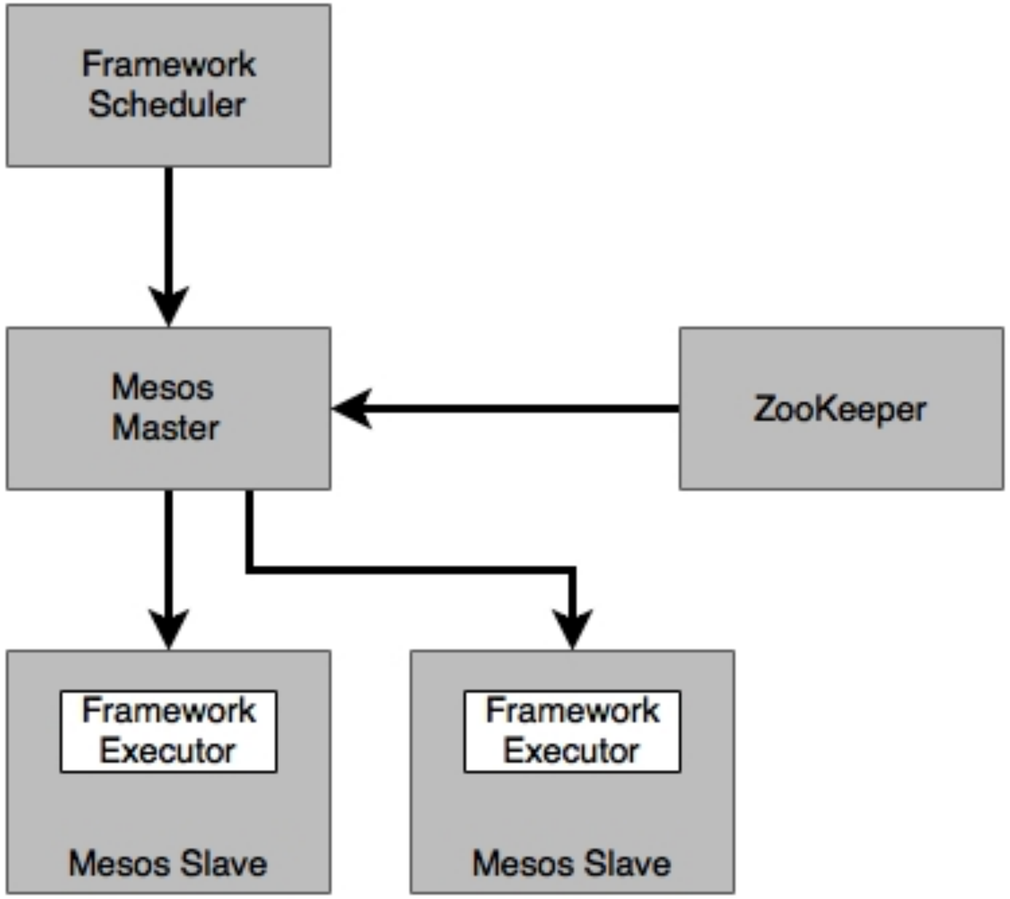
The Mesos architecture

- **ZooKeeper**: ZooKeeper is a centralized coordination server used in Mesos to coordinate activities across the Mesos cluster
  - Mesos uses ZooKeeper for leader election in case of a Mesos master failure



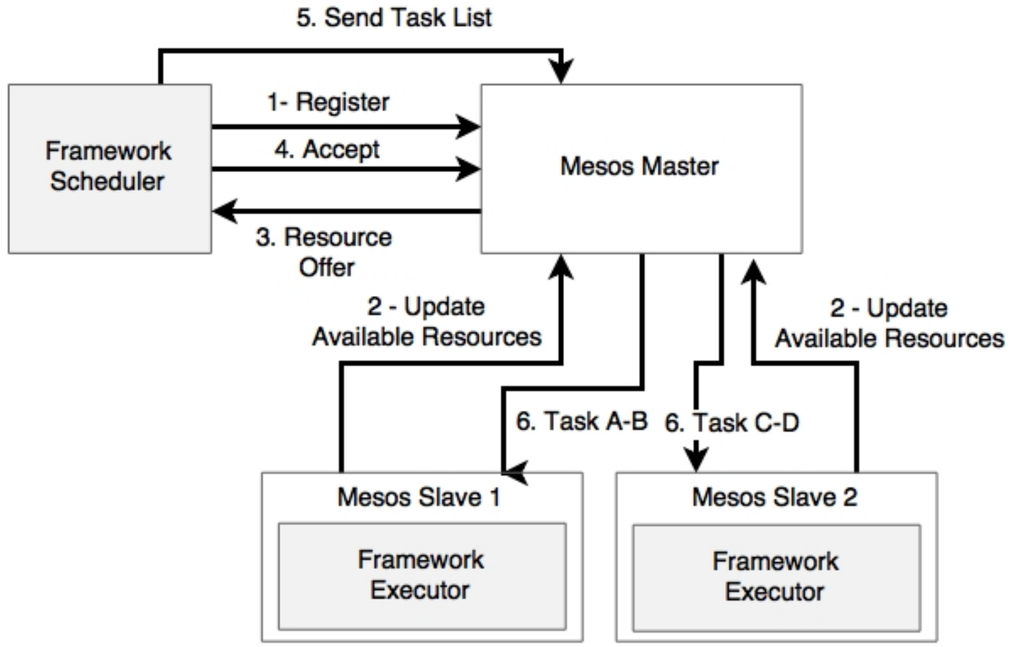
The Mesos architecture

- **Framework**: The Mesos framework is responsible for understanding the application's constraints, accepting resource offers from the master, and finally running tasks on the slave resources offered by the master
  - The Mesos framework consists of two components: the framework scheduler and the framework executor:
    - The scheduler is responsible for registering to Mesos and handling resource offers
    - The executor runs the actual program on Mesos slave nodes



The Mesos architecture

- The framework is also responsible for enforcing certain policies and constraints
- For example, a constraint can be, let's say, that a minimum of 500 MB of RAM is available for execution.
- Frameworks are pluggable components and are replaceable with another framework.



The Mesos architecture

- Mesos supports a number of frameworks, such as:
  - Marathon and Aurora for **long-running** processes, such as web applications
  - Hadoop, Spark, and Storm for **big data** processing
  - Chronos and Jenkins for **batch scheduling**
  - Cassandra and Elasticsearch for **data management**
- We will use Marathon to **run** dockerized microservices

Marathon

- Marathon is one of the Mesos framework implementations that can run both container as well as noncontainer execution
- Marathon ensures that the service started with Marathon continues to be available even if the Mesos slave it is hosted on fails
- Marathon is written in Scala and is highly scalable
- Offers a UI as well as REST APIs to interact with him, such as the start, stop, scale, and monitoring applications
- Marathon's high availability is achieved by running multiple Marathon instances pointing to a ZooKeeper instance
  - One of the Marathon instances acts as a leader, and others are in standby mode

Marathon

- Some of the basic features of Marathon include:
  - Setting resource constraints
  - Scaling up, scaling down, and the instance management of applications
  - Application version management
  - Starting and killing applications
- Some of the advanced features of Marathon include:
  - Rolling upgrades, rolling restarts, and rollbacks
  - Blue-green deployments