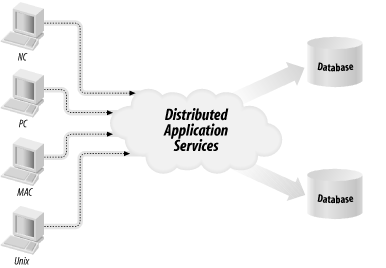
**Ce este o aplicatie distribuita?**

Distributed applications (distributed apps) are applications or software that runs on multiple computers within a network at the same time and can be stored on servers or with cloud computing. Unlike traditional applications that run on a single system, distributed applications run on multiple systems simultaneously for a single task or job.



Distributed apps can communicate with multiple servers or devices on the same network from any geographical location. The distributed nature of the applications refers to data being spread out over more than one computer in a network.

Distributed applications are broken up into two separate programs: the client software and the server software. The client software or computer accesses the data from the server or [cloud](http://searchnetworking.techtarget.com/definition/cloud) environment, while the server or cloud processes the data. Cloud computing can be used instead of servers or hardware to process a distributed application's data or programs. If a distributed application component goes down, it can[failover](http://searchstorage.techtarget.com/definition/failover) to another component to continue running.

Distributed applications allow multiple users to access the apps at once. Many developers, IT professionals or enterprises choose to store distributed apps in the cloud because ofcloud's [elasticity](http://searchcio.techtarget.com/definition/IT-elasticity) and scalability, as well as its ability to handle large applications or workloads.

**Suportul Haskell pentru dezvoltarea aplicatiilor distribuite. Platforma Cloud Haskell.**

Cloud Haskell is a set of libraries that bring Erlang-style concurrency and distribution to Haskell programs. This project is an implementation of that distributed computing interface, where processes communicate with one another through explicit message passing rather than shared memory.

Originally described by the joint [Towards Haskell in the Cloud](http://research.microsoft.com/en-us/um/people/simonpj/papers/parallel/remote.pdf) paper, Cloud Haskell has be re-written from the ground up and supports a rich and growing number of features for

* building concurrent applications using asynchronous message passing
* building distributed computing applications
* building fault tolerant systems
* running Cloud Haskell nodes on various network transports
* working with several network transport implementations (and more in the pipeline)
* supporting *static* values (required for remote communication)

Cloud Haskell comprises the following components, some of which are complete, others experimental.

* [distributed-process](https://github.com/haskell-distributed/distributed-process): Base concurrency and distribution support
* [distributed-process-platform](https://github.com/haskell-distributed/distributed-process-platform): The Cloud Haskell Platform - APIs
* [distributed-static](http://hackage.haskell.org/package/distributed-static): Support for static values
* [rank1dynamic](http://hackage.haskell.org/package/rank1dynamic): Like Data.Dynamic and Data.Typeable but supporting polymorphic values
* [network-transport](http://hackage.haskell.org/package/network-transport): Generic Network.Transport API
* [network-transport-tcp](http://hackage.haskell.org/package/network-transport-tcp): TCP realisation of Network.Transport
* [network-transport-inmemory](https://github.com/haskell-distributed/network-transport-inmemory): In-memory realisation of Network.Transport (incomplete)
* [network-transport-composed](https://github.com/haskell-distributed/network-transport-composed): Compose two transports (very preliminary)
* [distributed-process-simplelocalnet](http://hackage.haskell.org/package/distributed-process-simplelocalnet): Simple backend for local networks
* [distributed-process-azure](http://hackage.haskell.org/package/distributed-process-azure): Azure backend for Cloud Haskell (proof of concept)
* One of Cloud Haskell’s goals is to separate the transport layer from the process layer, so that the transport backend is entirely independent. In fact other projects can and do reuse the transport layer, even if they don’t use or have their own process layer (see e.g.[HdpH](http://hackage.haskell.org/package/hdph)).
* Abstracting over the transport layer allows different protocols for message passing, including TCP/IP, UDP, [MPI](http://en.wikipedia.org/wiki/Message_Passing_Interface), [CCI](http://www.olcf.ornl.gov/center-projects/common-communication-interface/), [ZeroMQ](http://zeromq.org/), [SSH](http://openssh.com/), MVars, Unix pipes, and more. Each of these transports provides its own implementation of the Network.Transport API and provide a means of creating new connections for use within Control.Distributed.Process.
* The following diagram shows dependencies between the various subsystems, in an application using Cloud Haskell, where arrows represent explicit directional dependencies.