# Cosmos Fundamentals

Author: Cosmos Team

Date: 6/3/2014

# What is Cosmos?

Cosmos is a platform for building Big Data applications.

There are two fundamental abilities that Cosmos provides as part of that platform:

* The ability to store massive amounts of data into the **Cosmos File System (CFS)**
* The power to process that data – by using our programming language called **Scope**

You may have heard of [**MapReduce**](http://en.wikipedia.org/wiki/Mapreduce). Cosmos implements its own form of MapReduce on CFS – which is a massively distributed file system.

# Value Proposition

You can unlock deep insights based on the data you already using Cosmos that let you be more effective in delivering your product, understanding your customers, and running your business.

How is Cosmos Different from other systems you may have heard of?

* Massive Scale. We support huge datasets (Petabyte range) and incredible amounts of computation.
* Cosmos makes working with Big Data much easier. Our Scope language allows someone with basic SQL and C# knowledge to process huge datasets without having to learn about MapReduce. Furthermore we have incredible integration with **Visual Studio**.
* Cosmos is also a **Shared Data Ecosystem** – Different teams can share their high-value datasets with each other.

How much does it Cost?

* Teams pay a fixed price for compute resources and storage every year.
* As Cosmos is a service, you don’t have to pay for maintaining and running your own machines – we provide the maintenance, availability, operations, etc.
* Though it won’t be obvious how now, Cosmos is all about “Scale at Low Cost” – we are able to provide our services by being very, very efficient with the resources we have.

# Massive

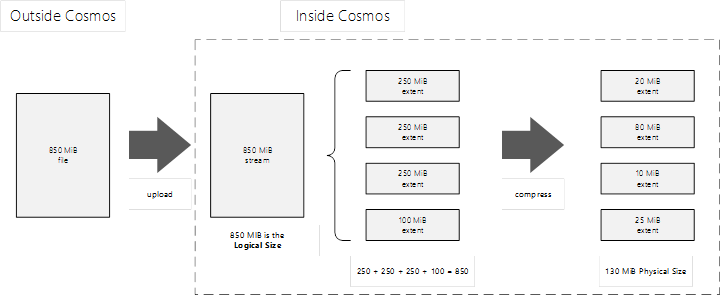
* Cosmos runs on about **110K machines** distributed over 4 different physical clusters.
* Cosmos stores about **1.2 Exabytes** of information.
* Cosmos runs about **95K Scope jobs** every day.

# How do users leverage Cosmos?

There are many nuanced uses such as query serving, search engine relevance, ads relevance, and so on, but the general Cosmos usage pattern follows:

* Process vast amounts of data ad-hoc or periodically
* Persist their data reliably for later use
* Share data between groups and gain access to valuable data from other groups

# How Does Cosmos Store Data?



When you upload or create a file in Cosmos we call it a **stream**. We first divide it into **extents**. The division of files into extents and how that division is done has **a dramatic impact upon the performance and reliability**.

Extents can be as big as 250 MiB, but no greater. Cosmos will compress extents so that it requires less physical disk space. So if you have a stream that is 850 MiB it may only take up 130 MiB “on disk” in Cosmos. This compression is under the covers and nothing you need to think about when you use Cosmos.

For reliability, Cosmos stores three copies of an extent and distributes those copies across machines. In this way, Cosmos achieves very high reliability for your data. If one machine that stores data goes down, then there two other machines available with copies for that extent.

## The Creation of Streams

Streams are created via two mechanisms:

* Upload from machines outside of Cosmos
* Scope Jobs create streams as output

## Uploading Data

Depending on the scenarios there are different tools available.

### For simple interactive uploads:

* Cosmos Portal
* Command-Line tools such as Cosmos PowerShell or scope.exe

### For programmatic uploads:

* Cosmos PowerShell
* Directly use the VcClient.VC.Upload API

### Bulk scenario:

Imagine you have 50 machines and each is generating data. You want a single stream that has all the data from all the machines.

Now, data upload can occur from systems running in the PHX domain, as well as from systems in other corporate domains. A common case is for logs generated by serving layers of an online service to be uploaded into Cosmos. Generally, the pattern used is to run a lightweight upload utility on the serving machines, which periodically pushes the logs into Cosmos. Other systems may aggregate logs from many serving machines before uploading to Cosmos. Cosmos provides a utility called Cosmos Data Loader, which is used by many teams to do this upload. Other utilities have been developed by the community using the publicly available APIs in the Cosmos VCClient library. Data can also be uploaded using the Scope.exe command-line utility.

Once in Cosmos, data is compressed as shown above. Neither the replication nor the compression is exposed to the user; the view of data in Cosmos is of a traditional file system. One should note that a stream is an append-only container for data. Once a stream is completely written, it becomes immutable and can only be read from or deleted. Streams are composed of units of data called “extents.” The extent is the unit of replication of data by the system, and the unit guaranteed to be processed serially by SCOPE programs. Users are not exposed to the concept of extents – users interact with Cosmos exclusively at the level of streams. However, an understanding of extents and details of uploading can be important when designing data processing pipelines, particularly when the data involved is not simply text data. Data in Cosmos can be read and processed by SCOPE jobs. Some data may be generated in Cosmos and used from within Cosmos without ever being taken outside of the system. Data can downloaded from Cosmos using the Scope.exe command line utility or by writing a program leveraging the VCClient API. Cosmos Interactive Queries provide another means for extracting data from Cosmos. A common pattern is for SCOPE jobs to run over very large datasets in Cosmos, producing new, smaller, streams containing aggregations or filtered views of data from the larger dataset. This smaller stream is then downloaded from Cosmos and may be loaded into Excel or other tools for analysis.

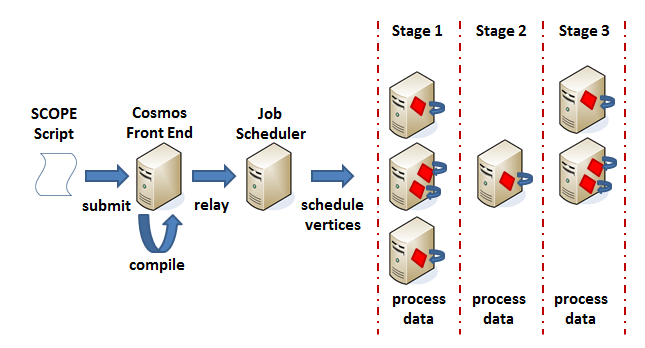
# How do users program and process data in Cosmos?

Users program using SCOPE and it is Cosmos' custom language. SCOPE is similar to SQL so it easy to program with, but SCOPE also comes with C# functionality, so SCOPE is powerful and versatile as well. Users develop in Visual Studio 2010 using the SCOPE plug-in that we provide and, after they've written a script, they submit it using the IDE to our Front-End, which hands it off to Cosmos for processing. The SCOPE language is the 3rd generation of programmatic interface to the Cosmos system, and SCOPE is designed to be easy enough to use for simple analysis that non-programmers with a familiarity with SQL can easily pick it up. However, SCOPE also provides the power to create very complex applications with high performance requirements.

SCOPE development generally begins with authoring a script in the Visual Studio plug-in (referred to moving forwards as simply the “IDE” or Integrated Development Environment). The IDE provides a local compile functionality which lets you check the script for syntax and other errors before submitting. The IDE also provides functionality to run the SCOPE program locally on your development machine. This simplified execution mode roughly simulates what happens on the cluster during execution, and helps to validate results and discover bugs much more quickly than waiting for these to be caught during cluster execution.

Using the IDE, local runs can be debugged with step-through debugging very similar to the step-through debugging provided for other languages in Visual Studio. Cluster runs cannot be debugged in this manner. However, when a specific vertex in a job fails, causing the job itself to fail, then the IDE provides the ability to replicate the execution of that vertex locally, again allowing step-through debugging of the specific code running on that vertex.

# How does Cosmos process data?



As mentioned above, users submit scripts in Visual Studio 2010 and the Cosmos Front-End receives the script, compiles it for errors, and then optimizes the script. Afterwards, Cosmos' execution engine will break down the script's commands into a series of tasks that are delegated to vertices. These vertices, essentially a processing token, are run on many machines in parallel where the necessary data resides, and then the vertices write the output data--some output data is consumed by other vertices during the job's run-time, known as intermediate data, and the other type is global data, meaning we persist it with 2 or 3 copies for later use.

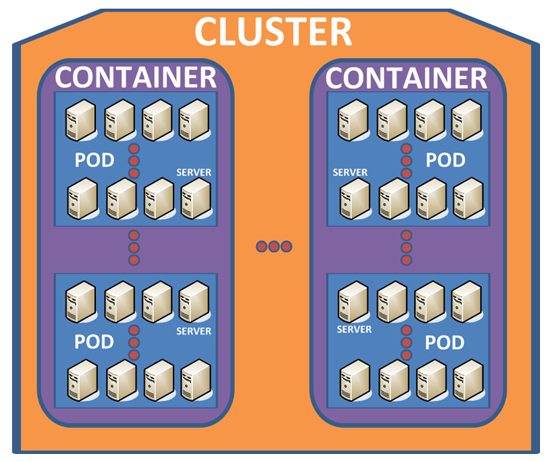
The diagram above illustrates what a job is at a high level, and this job above must be associated with a Virtual Cluster (VC) to be submitted. So, a user, for example, will submit credentials when their job is submitted along with a target VC (the VC the user will occupy processing and storage resources on), and then Cosmos will begin running the job on the target VC.

In the following section, we will clarify the VC concept and how it is used for easy data access, in addition to controlling user storage and processing quotas; but, for now, it is sufficient to understand that every user must be associated with a VC to interact with Cosmos, and that the user's VC dictates his or her storage restrictions as well as processing resources. For example, let’s take a hypothetical VC named "foo" with a size of 100. This would mean that at any given point in time this "foo" is guaranteed 100 nodes to run jobs on. But (of course), it’s not really that simple. What does “size” mean, and what do “nodes” mean in this context? For Cosmos, nodes refer to a machine – in fact, any machine – and there can be multiple vertices running per machine. Therefore, in the image above, the job has four vertices running in Stage 1, one in Stage 2, and three in Stage 3. To solidify the details: "foo" may be guaranteed 100 tokens meaning 100 high priority vertices are gauranteed to the VC, but not all 100 may be used because the jobs running that are associated with the VC may not be able to use more parallelism (more vertices won't help), or the VC is not being completely used up--users can dictate % VC allocation and maybe "foo" has 4 jobs running with 10% allocation so 60% of the VC's resources are still free.

The VC size is a minimal guarantee only. If there are unused computation resources in the cluster, and a job running in a VC is using the entire allocation of that VC and could use even more resources, then Cosmos will give the unused resources to that job. If jobs start in other VCs such that they need resources to meet their VC size guarantee, then Cosmos will reclaim the necessary execution resources to give to the “needy” VC. This reclamation, however, is not generally done by stopping vertices that are already executing. Rather, once a vertex is complete the physical resources become balanced back to ensure each VC has the guaranteed execution capacity. An implication of this is that the minimal execution capacity guarantee of a VC is rather loose; it is not necessarily the case that there will instantaneously be that number of resources available – sometimes there may be a delay while vertices from other VCs finish such that the execution resources can become available to fulfill the resource guarantee.

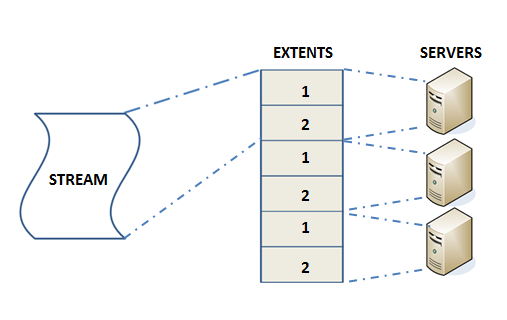
# How is my data organized?

Cosmos consists of three clusters, each cluster with eight-to-ten containers, each container with forty pods, and each pod with forty-five servers.



The above describes the physical layout; however, you will interact with Cosmos using virtual directories known as a Virtual Cluster (VC). Each user is associated with a VC and there are resources (storage and processing quotas) allocated specifically to a VC, and each VC has a queue as well. The motivation behind virtualizing directories was to make everything easier for the user. So, when the user interacts with Cosmos, the user will see

/dir1/dir2/STREAM.txt



which is actually many extents strewn across the Cosmos Cluster of thousands of machines. The stream does not exist on the machines as a single entity anymore, but we allow users to interact with their data as if it is a single file.

# Appendix

## Terminology

* Virtual cluster (VC): A virtual directory that users manage and use to interact with Cosmos
* Scope: Cosmos' custom language for processing data
* Job: A single SCOPE program that runs as a distributed computation using VC directories and resources
* Extent: Cosmos' unit of storage
* Stream: Immutable files stored in a VC that users read from to process data
* Vertex: Cosmos' unit of execution
* Node: A machine in Cosmo
* Token: A contractual guarantee that we will provide a high priority vertex. One token guarantees a single high priority vertex

# Change History

|  |  |
| --- | --- |
| Version | Description |
| YYYY-MM-DD | Description |
| 2011-12-13 | Initial Version |
| 2014-06-03 | Proofread and fixed errors |