Ceph Essentials

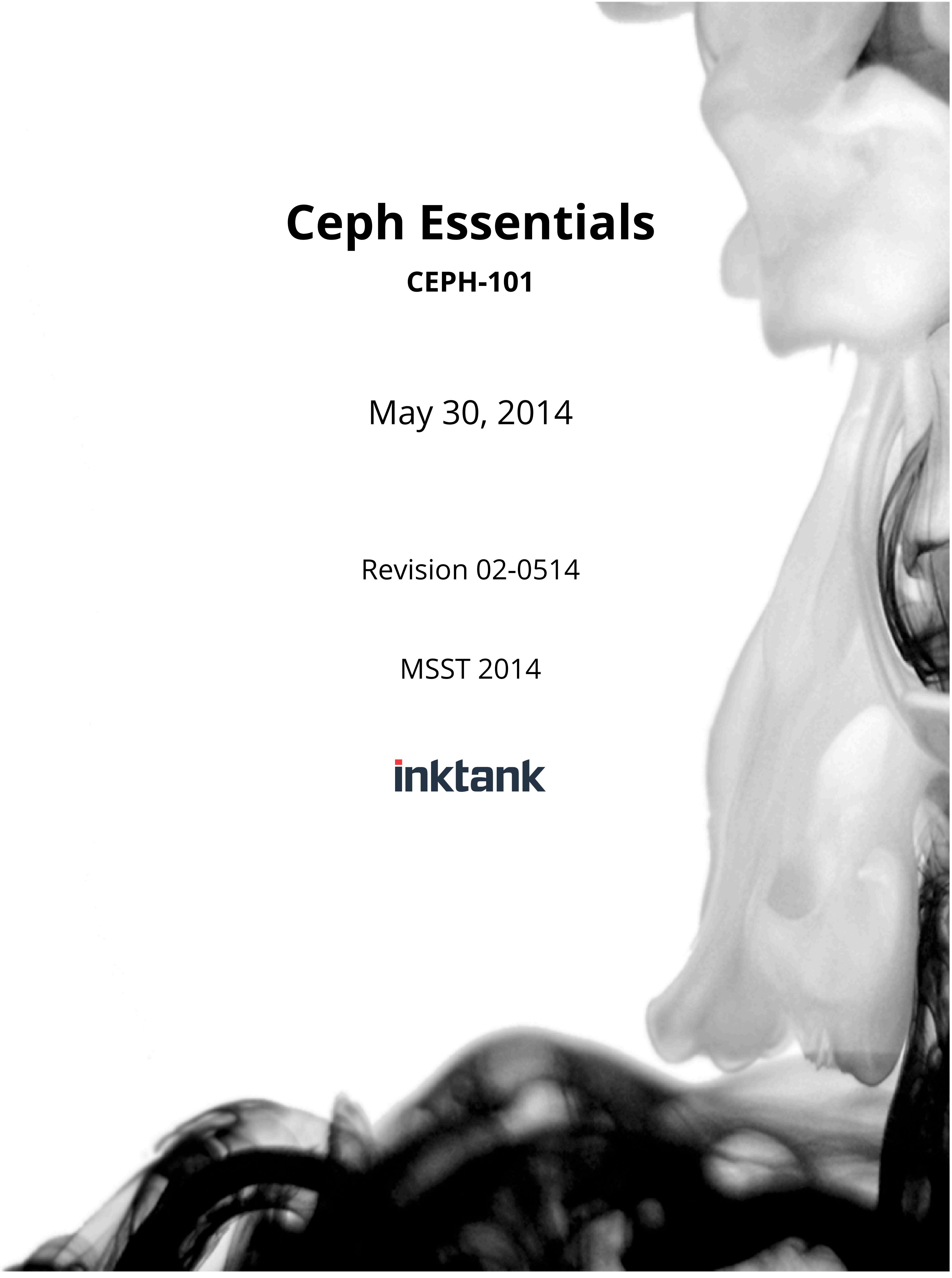
CEPH-101

May 30, 2014

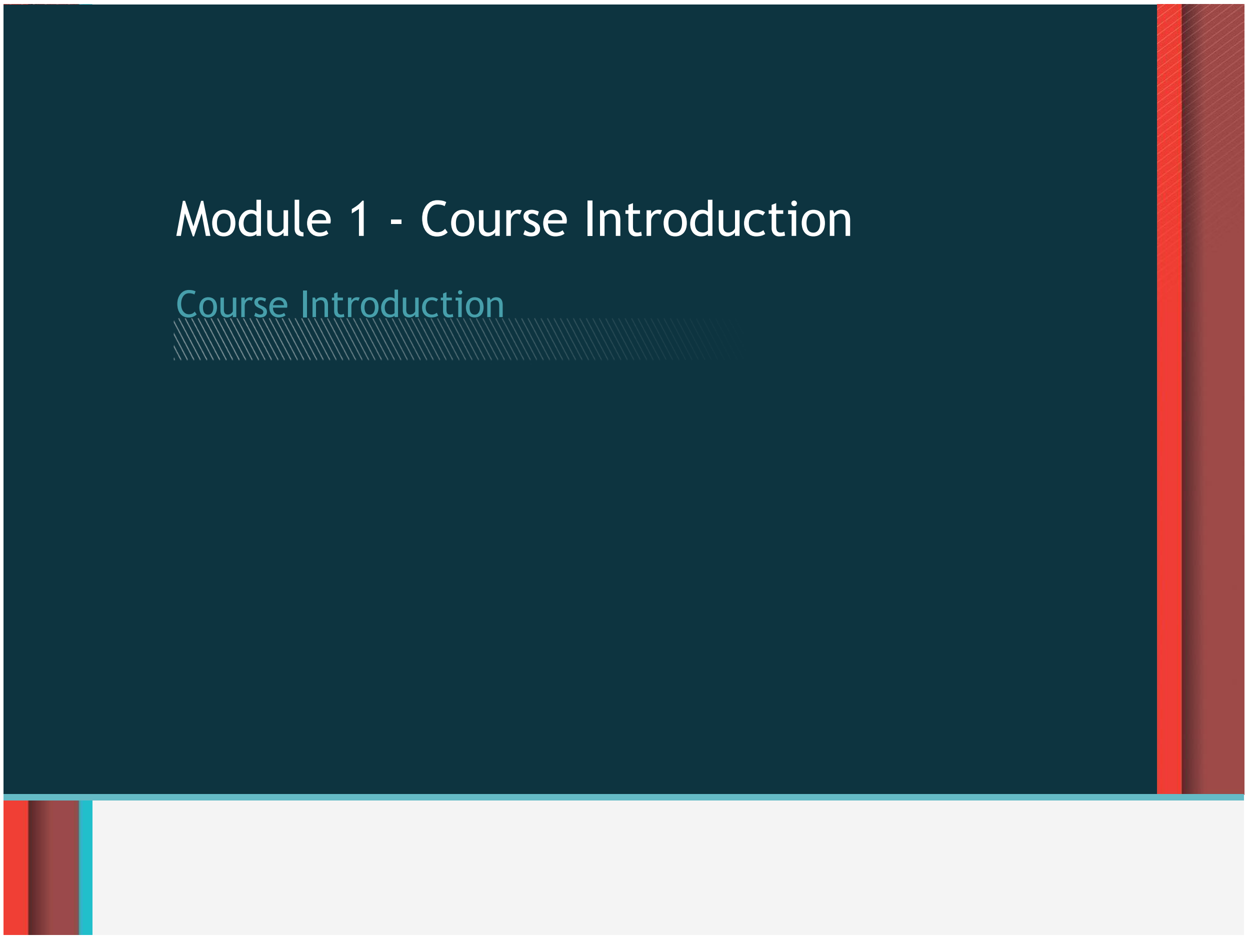
Revision 02-0514

MSST 2014

inktank

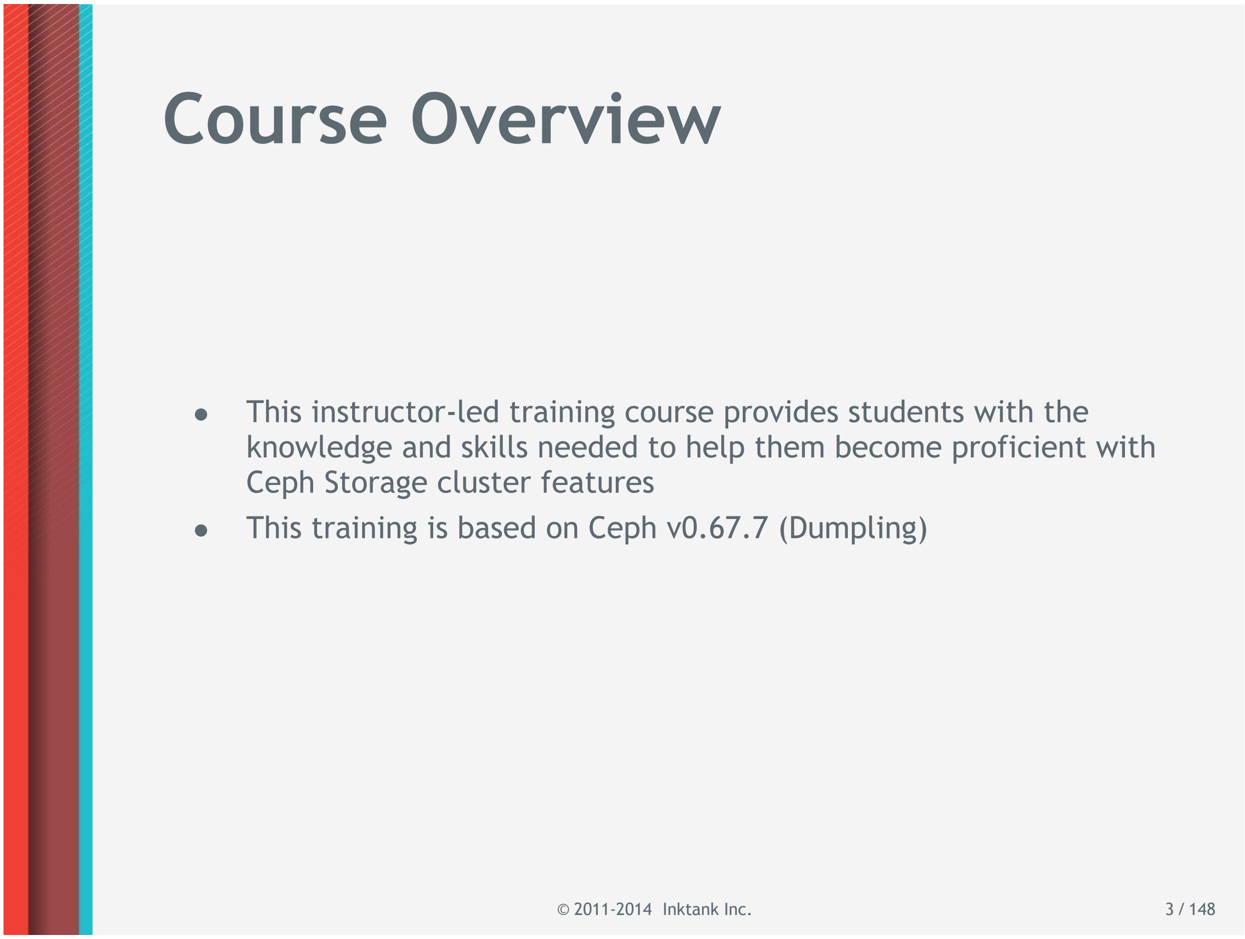


COURSE OUTLINE



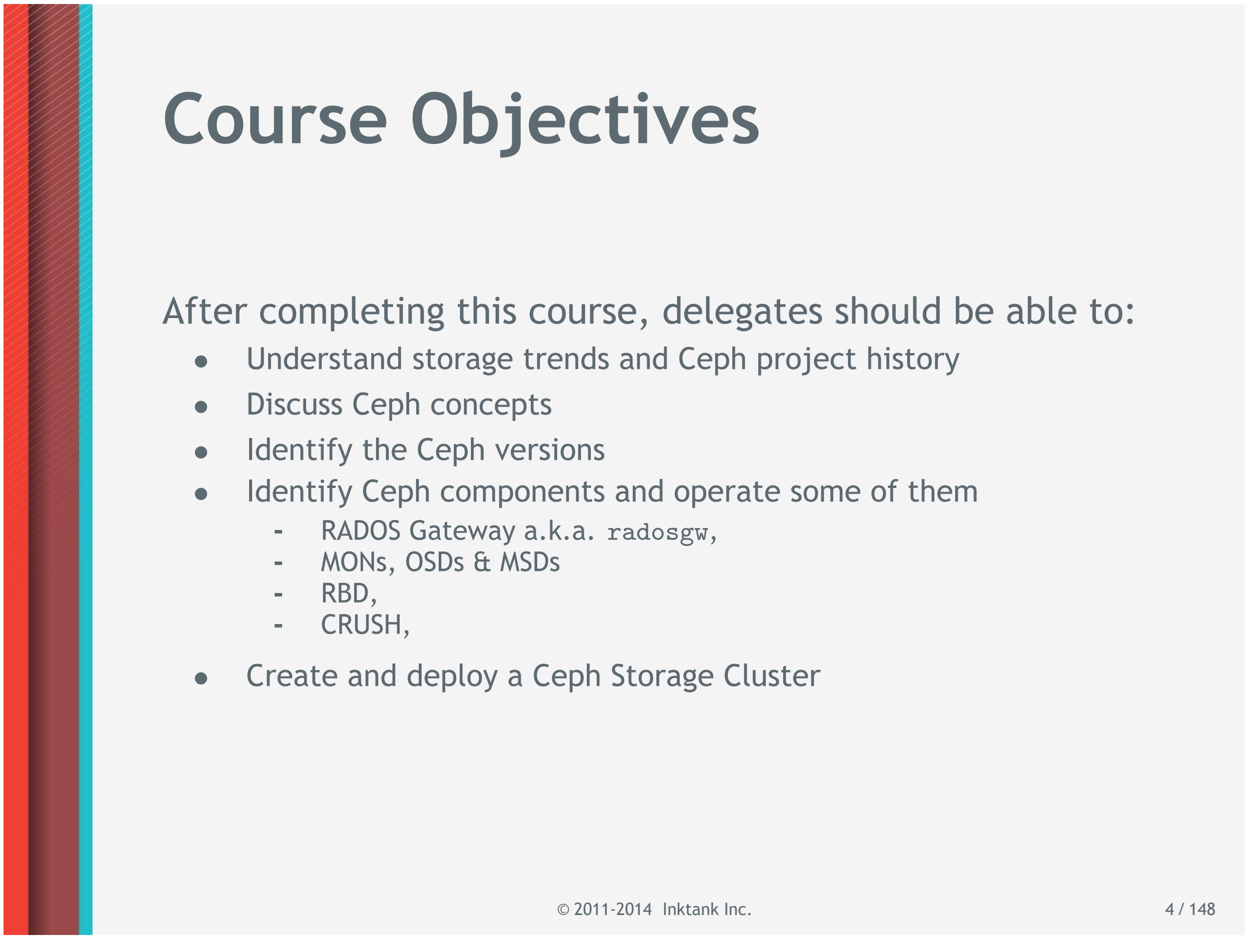
Module 1 - Course Introduction

Course Introduction



Course Overview

- This instructor-led training course provides students with the knowledge and skills needed to help them become proficient with Ceph Storage cluster features
- This training is based on Ceph v0.67.7 (Dumpling)



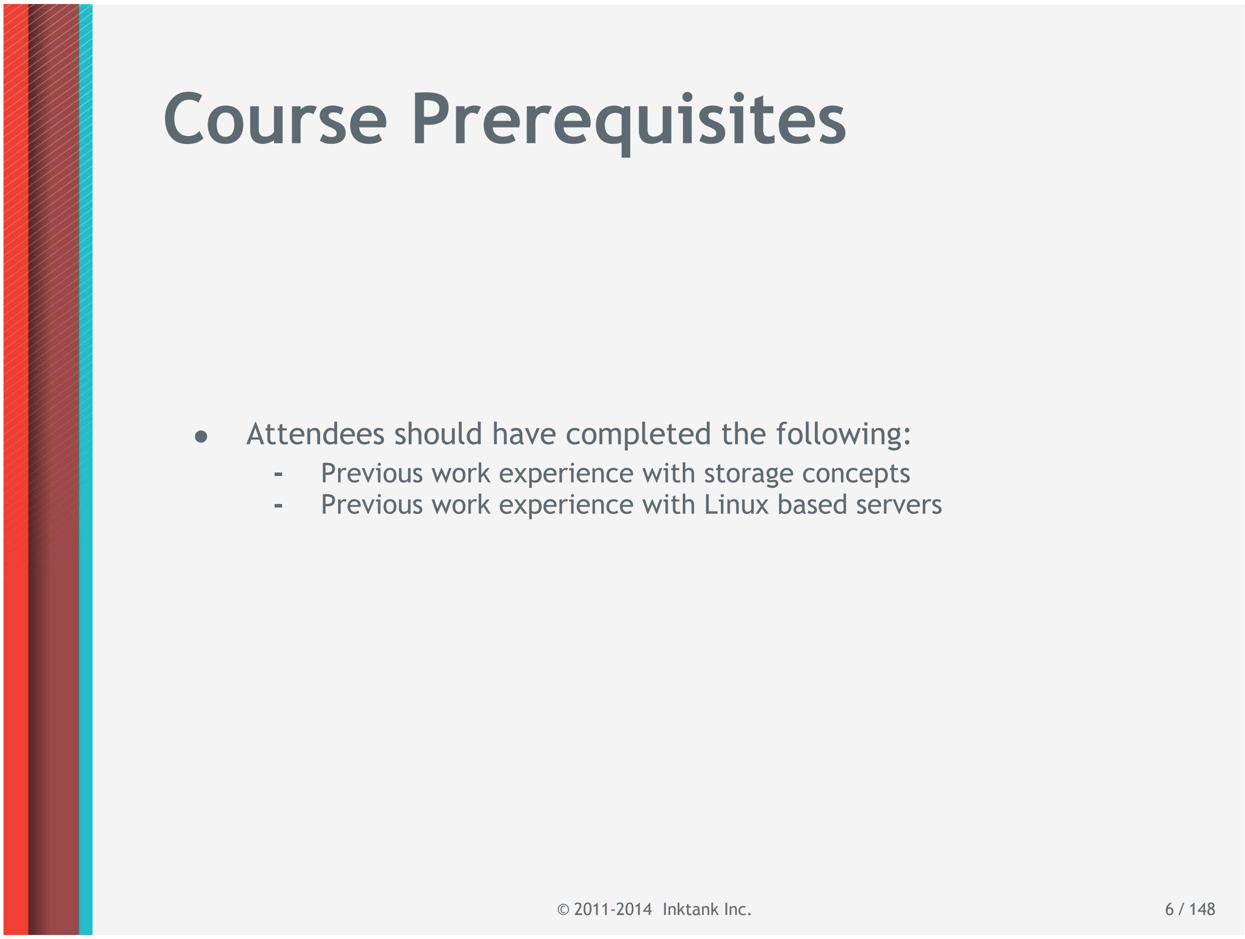
Course Objectives

After completing this course, delegates should be able to:

- Understand storage trends and Ceph project history
- Discuss Ceph concepts
- Identify the Ceph versions
- Identify Ceph components and operate some of them
  - RADOS Gateway a.k.a. `radosgw`,
  - MONs, OSDs & MSDs
  - RBD,
  - CRUSH,
- Create and deploy a Ceph Storage Cluster

Course Agenda

- Module 1 - Course Introduction
- Module 2 - Ceph History and Components
- Module 3 - Ceph Data Placement
- Module 4 - RADOS Object Store
- Module 5 - Ceph Block Storage (Ceph RBD)
- Module 6 - Ceph File systems (CephFS)
- Module 7 - Creating a Ceph Storage Cluster
    - LAB ; Deploying your cluster with ceph-deploy



Course Prerequisites

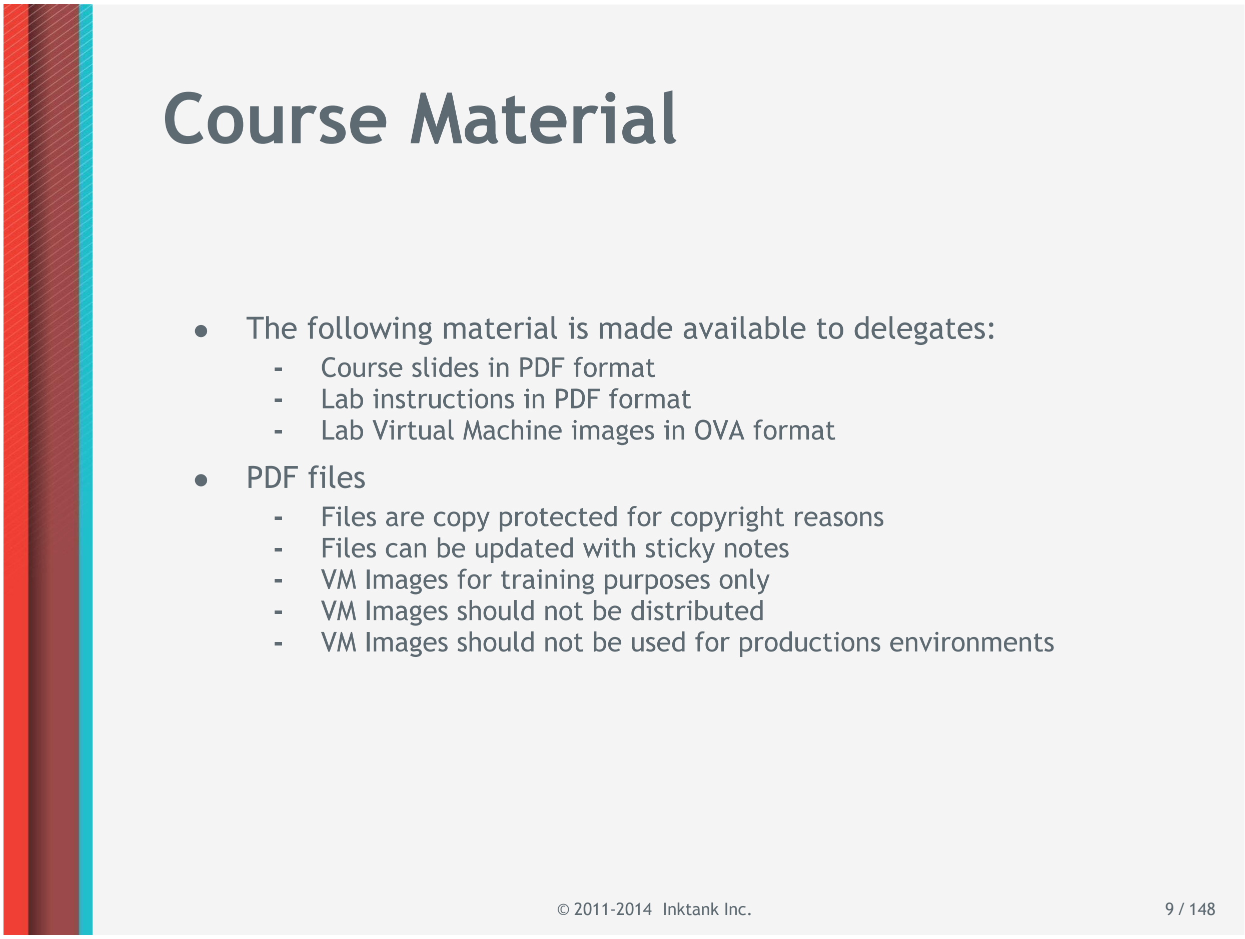
- Attendees should have completed the following:
  - Previous work experience with storage concepts
  - Previous work experience with Linux based servers

Course Catalog

- The following training are available:
  - CEPH-100 Ceph Fundamentals (ILT)
  - CEPH-101 Ceph Essentials (WBT)
  - CEPH-110 Ceph Operations & tuning (ILT & VCT)
  - CEPH-120 Ceph and OpenStack (ILT & VCT)
  - CEPH-130 Ceph Unified Storage for OpenStack (VCT)
  - CEPH-200 Ceph Open Source Development (ILT)

Course Material

- The following material is made available to delegates:
  - Course slides in PDF format
  - Lab instructions in PDF in OVA format
  - VM Images are for training purposes only
  - VM Images should not be distributes
  - VM Images should not be used for production environments



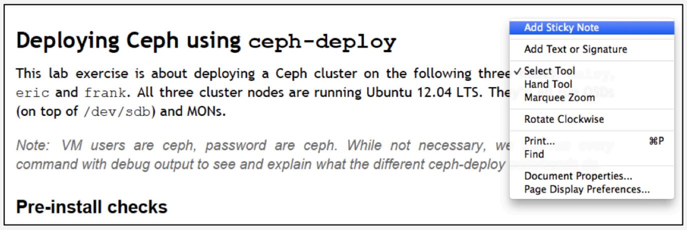
Course Material

- The following material is made available to delegates:
  - Course slides in PDF format
  - Lab instructions in PDF format
  - Lab Virtual Machine images in OVA format
- PDF files
  - Files are copy protected for copyright reasons
  - Files can be updated with sticky notes
  - VM Images for training purposes only
  - VM Images should not be distributed
  - VM Images should not be used for productions environments

Course Material

How to add a note to your PDF files

- Right click in the PDF file



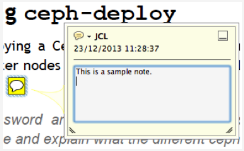
Deploying Ceph using `ceph-deploy`

This lab exercise is about deploying a Ceph cluster on the following three
`eric` and `frank`. All three cluster nodes are running Ubuntu 12.04 LTS. The
(on top of `/dev/sdb`) and MONs.

Note: VM users are ceph, password are ceph. While not necessary, we
command with debug output to see and explain what the different ceph-deploy

Pre-install checks

- From the pop up menu select ‘Add Sticky Note’



g `ceph-deploy`

ying a Ce
er nodes

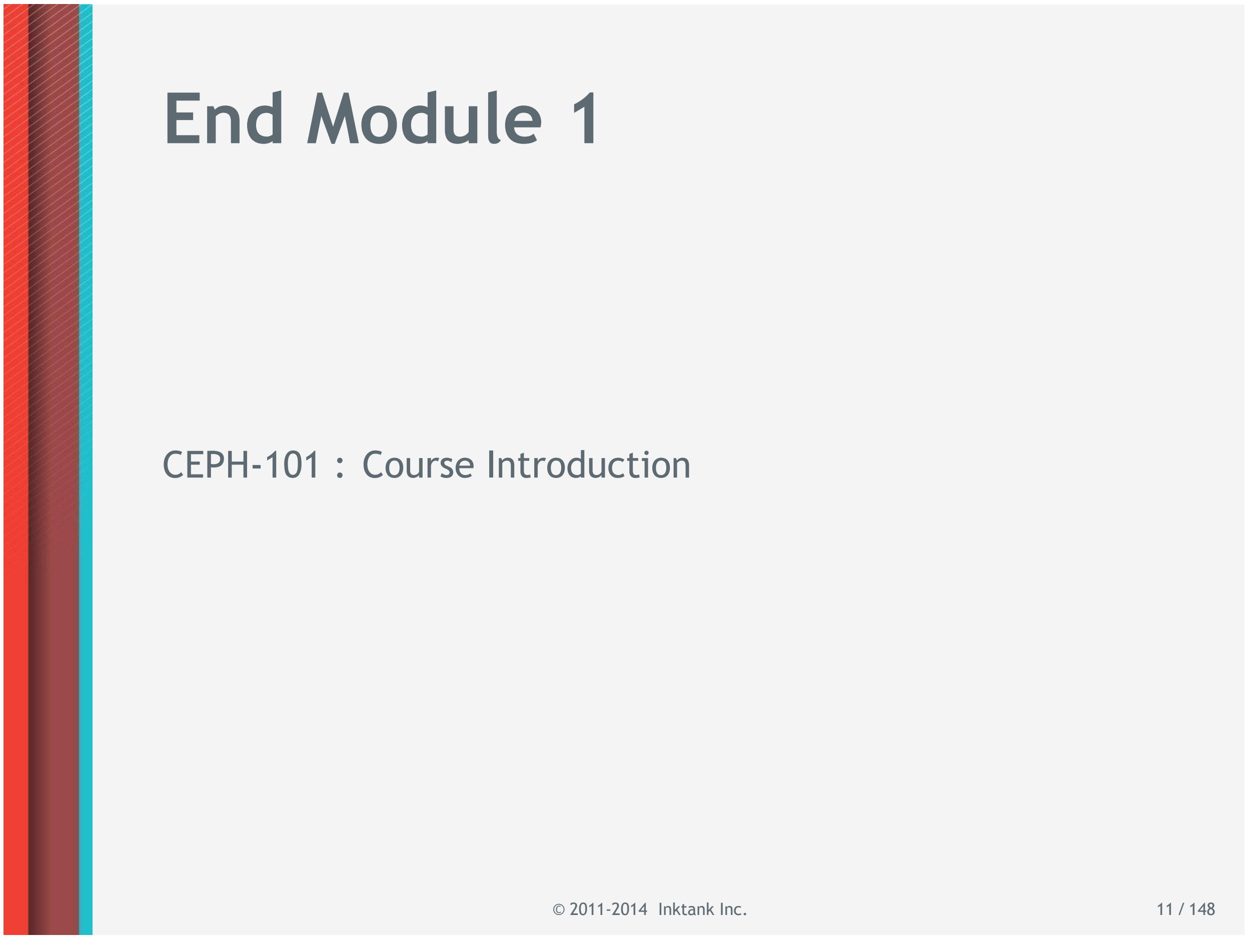
sword ar
e and explain what the different ceph

This is a sample note.

Shortcut is CTRL+6 on Microsoft Windows™

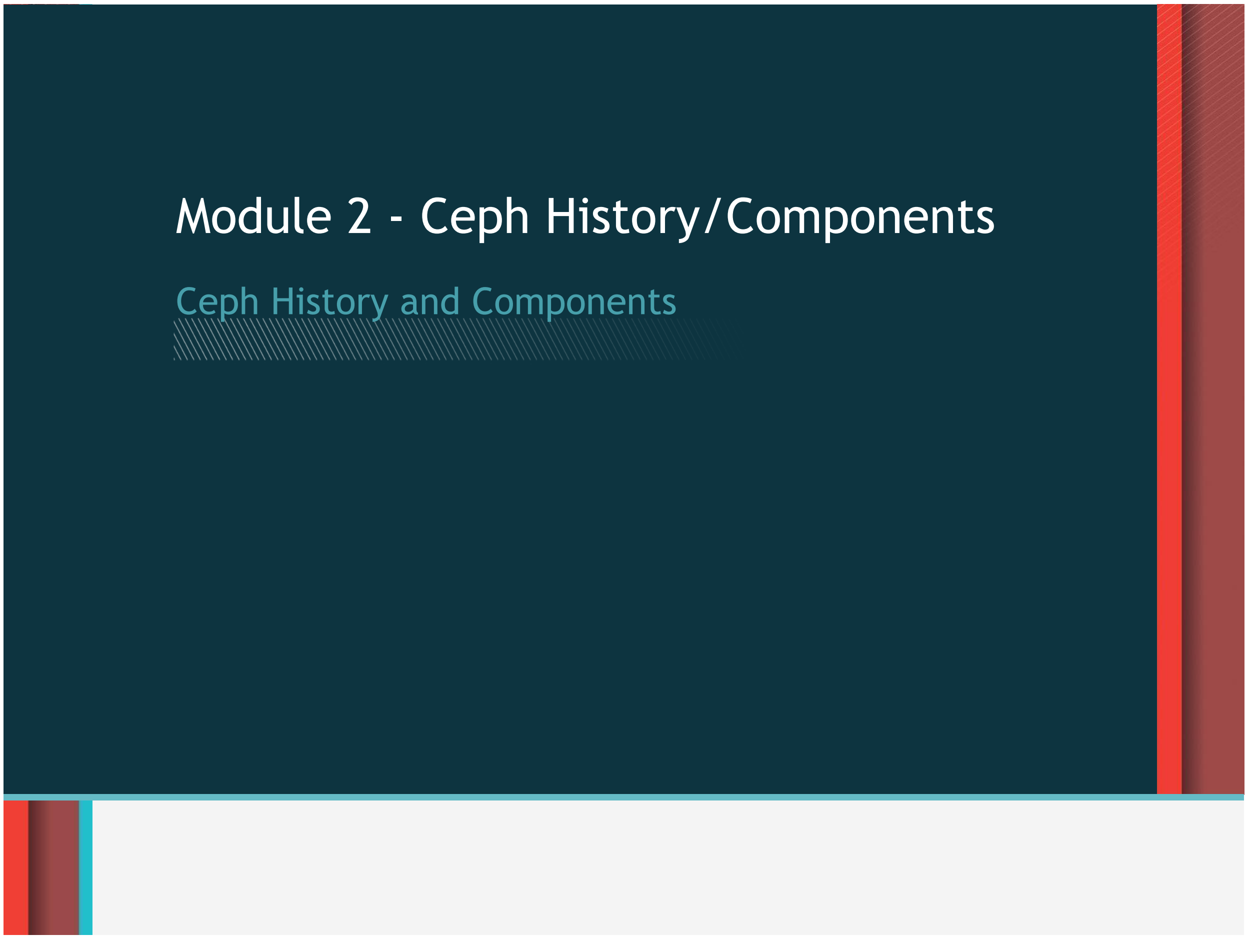
Shortcut is CMD+6 on Apple OSX™

Do not forget to save your file



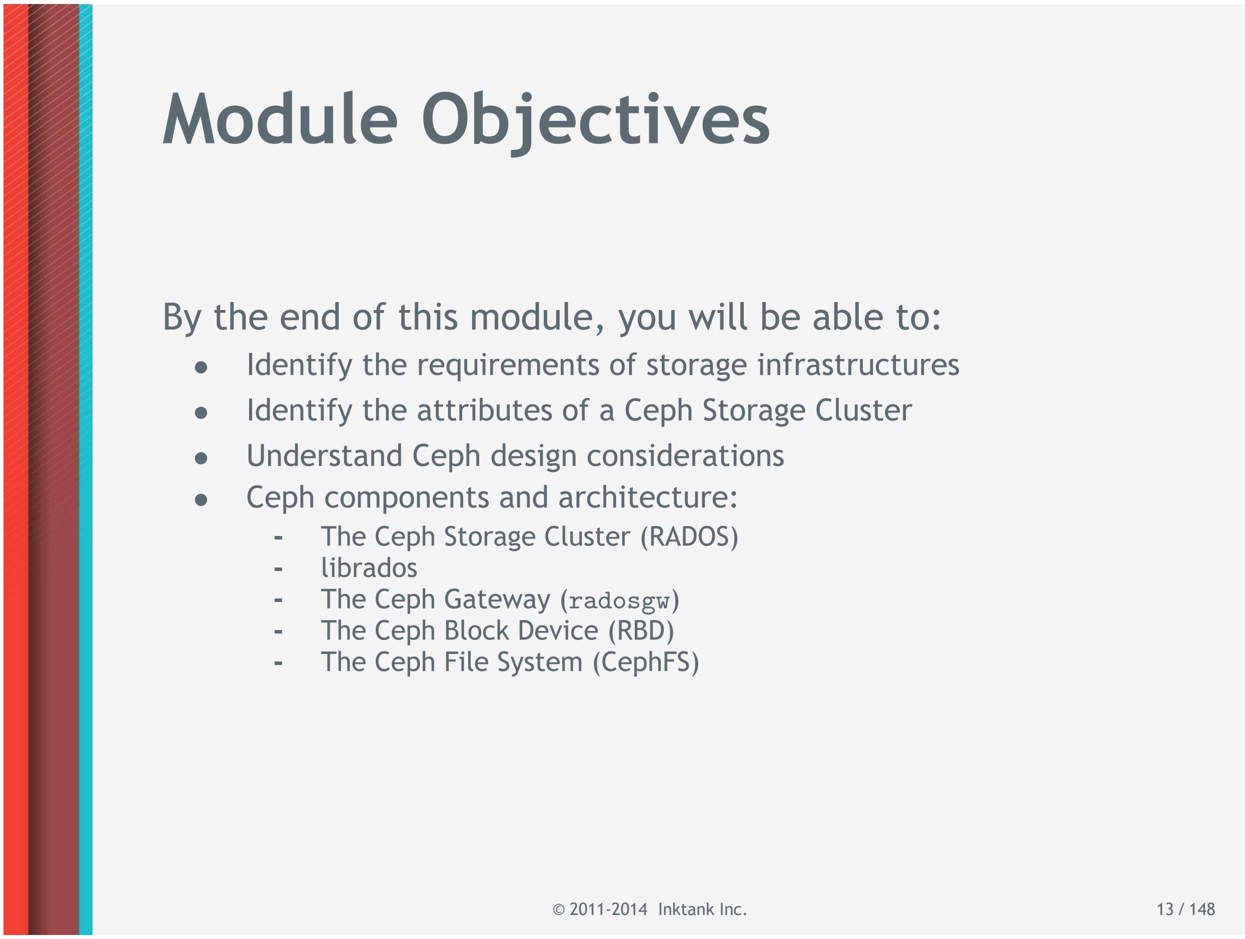
End Module 1

CEPH-101 : Course Introduction



Module 2 - Ceph History/Components

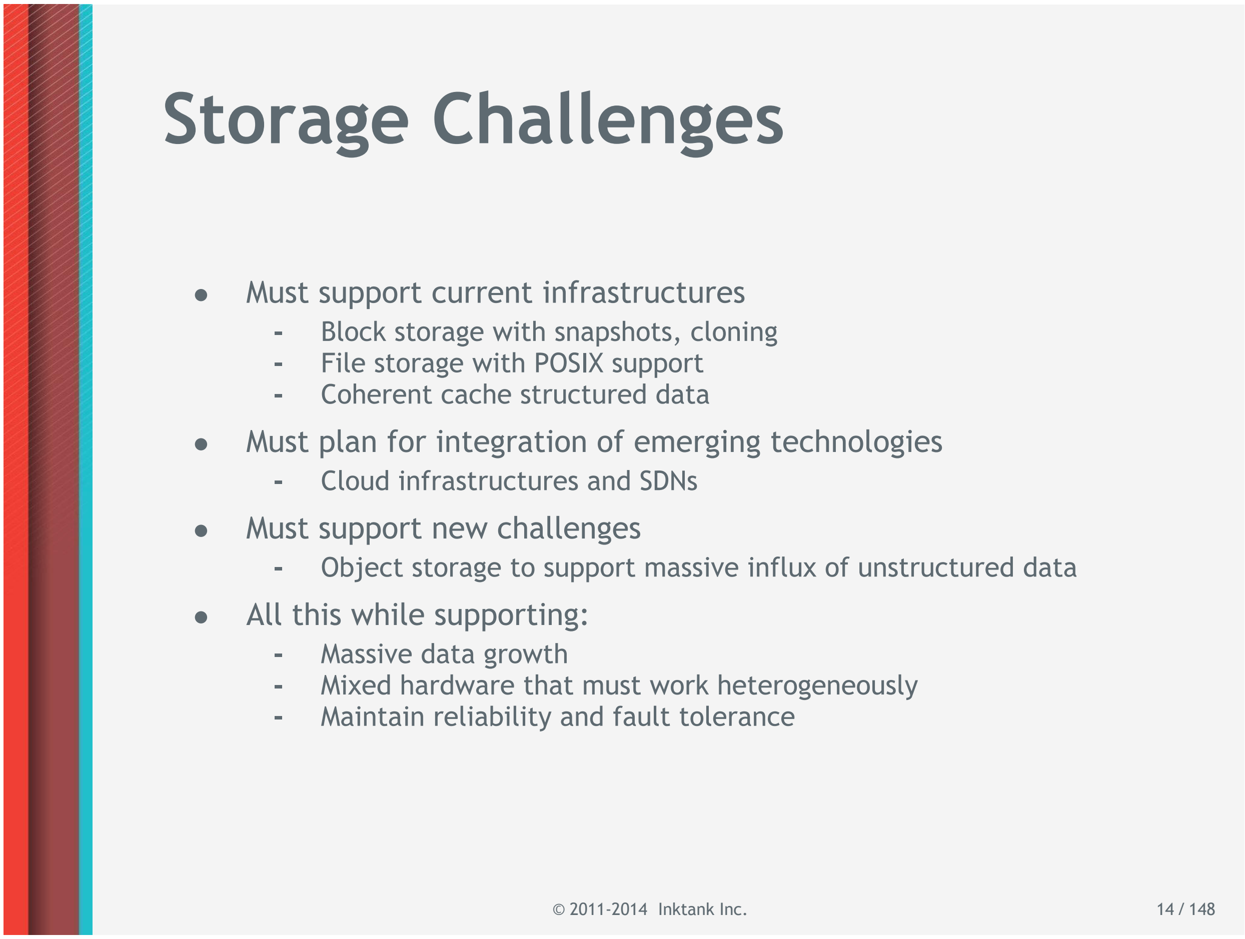
Ceph History and Components



Module Objectives

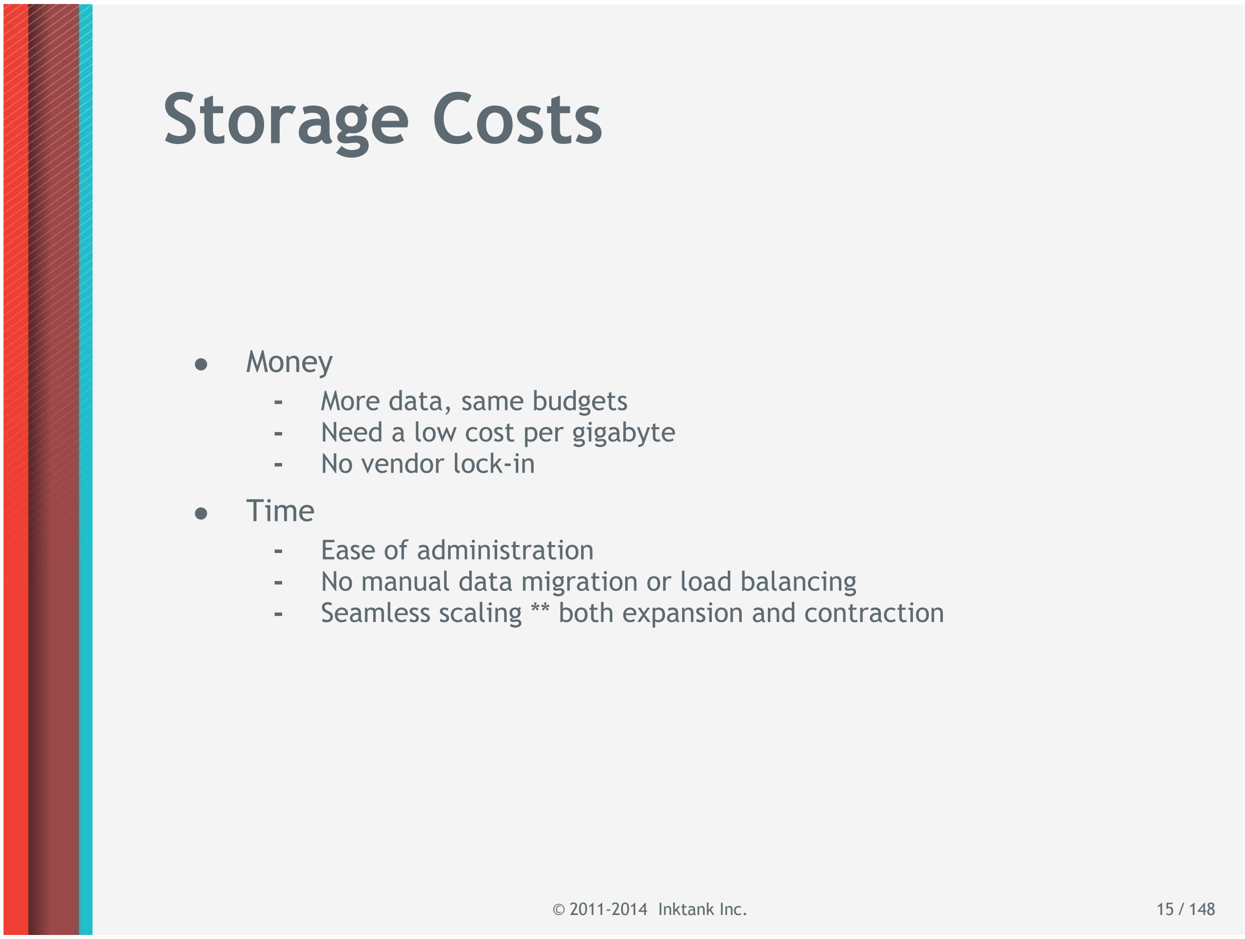
By the end of this module, you will be able to:

- Identify the requirements of storage infrastructures
- Identify the attributes of a Ceph Storage Cluster
- Understand Ceph design considerations
- Ceph components and architecture:
  - The Ceph Storage Cluster (RADOS)
  - librados
  - The Ceph Gateway (`radosgw`)
  - The Ceph Block Device (RBD)
  - The Ceph File System (CephFS)



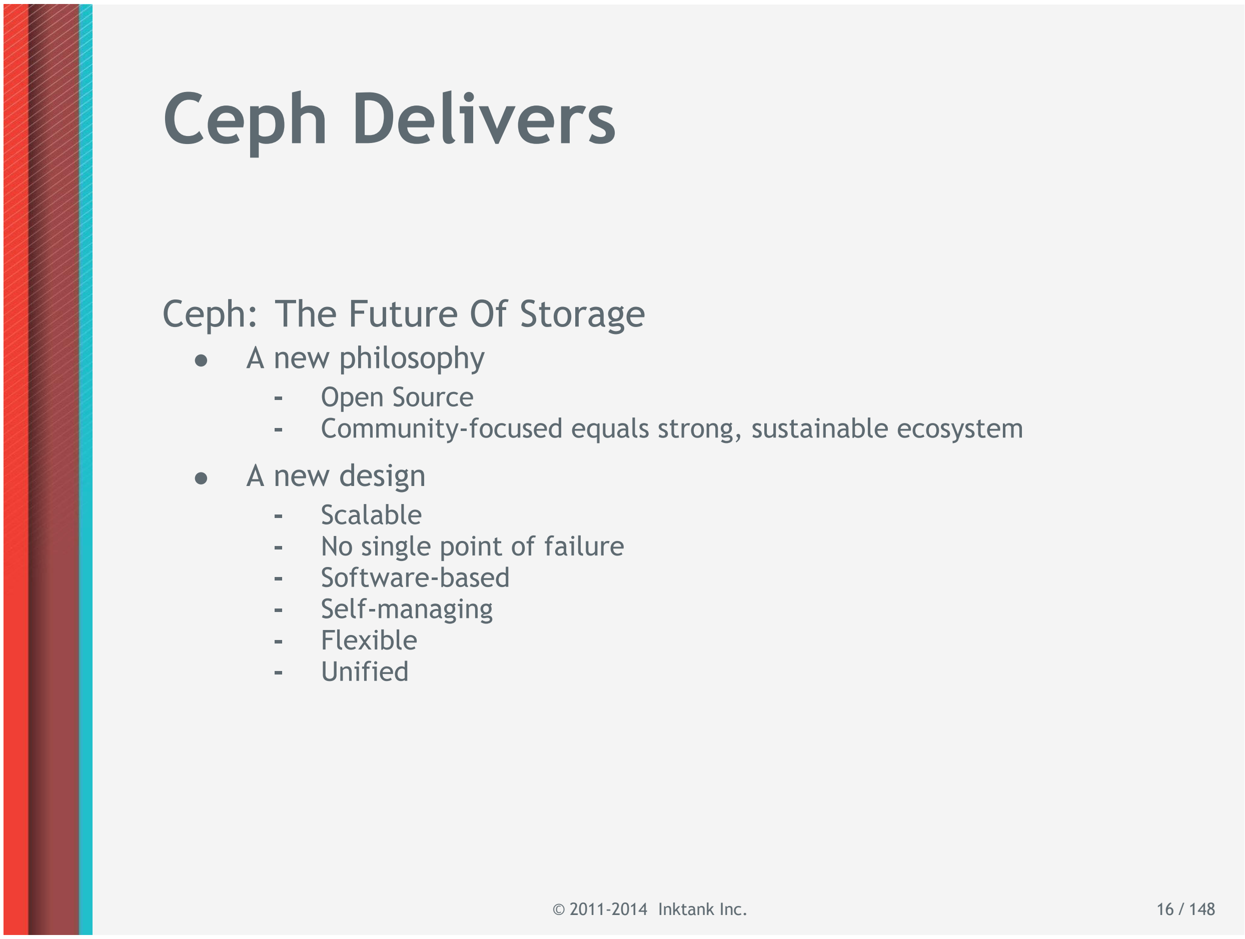
Storage Challenges

- Must support current infrastructures
  - Block storage with snapshots, cloning
  - File storage with POSIX support
  - Coherent cache structured data
- Must plan for integration of emerging technologies
  - Cloud infrastructures and SDNs
- Must support new challenges
  - Object storage to support massive influx of unstructured data
- All this while supporting:
  - Massive data growth
  - Mixed hardware that must work heterogeneously
  - Maintain reliability and fault tolerance



Storage Costs

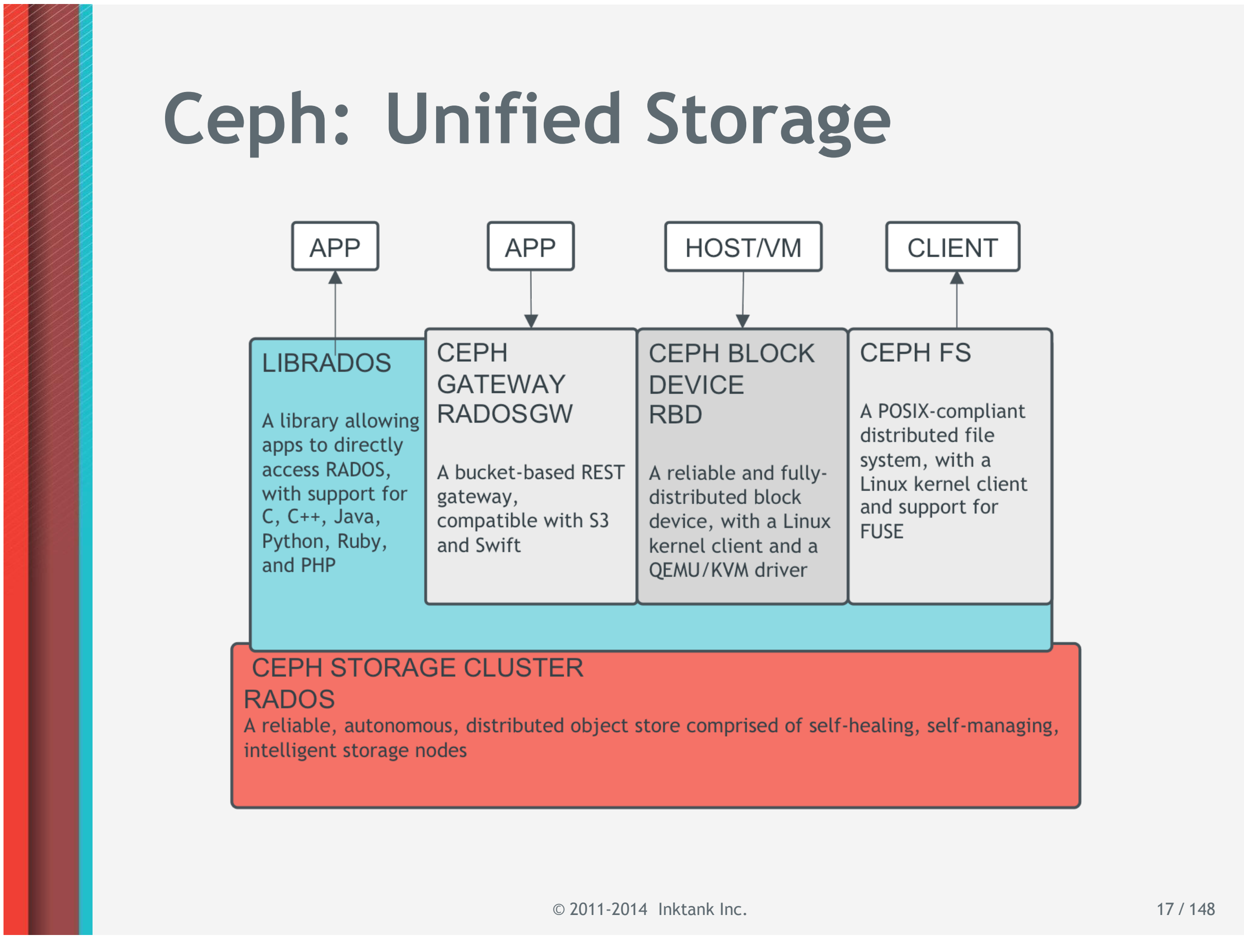
- Money
  - More data, same budgets
  - Need a low cost per gigabyte
  - No vendor lock-in
- Time
  - Ease of administration
  - No manual data migration or load balancing
  - Seamless scaling ** both expansion and contraction



Ceph Delivers

Ceph: The Future Of Storage

- A new philosophy
  - Open Source
  - Community-focused equals strong, sustainable ecosystem
- A new design
  - Scalable
  - No single point of failure
  - Software-based
  - Self-managing
  - Flexible
  - Unified



Ceph: Unified Storage

APP

APP

HOST/VM

CLIENT

LIBRADOS

A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP

CEPH GATEWAY RADOSGW

A bucket-based REST gateway, compatible with S3 and Swift

CEPH BLOCK DEVICE RBD

A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver

CEPH FS

A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE

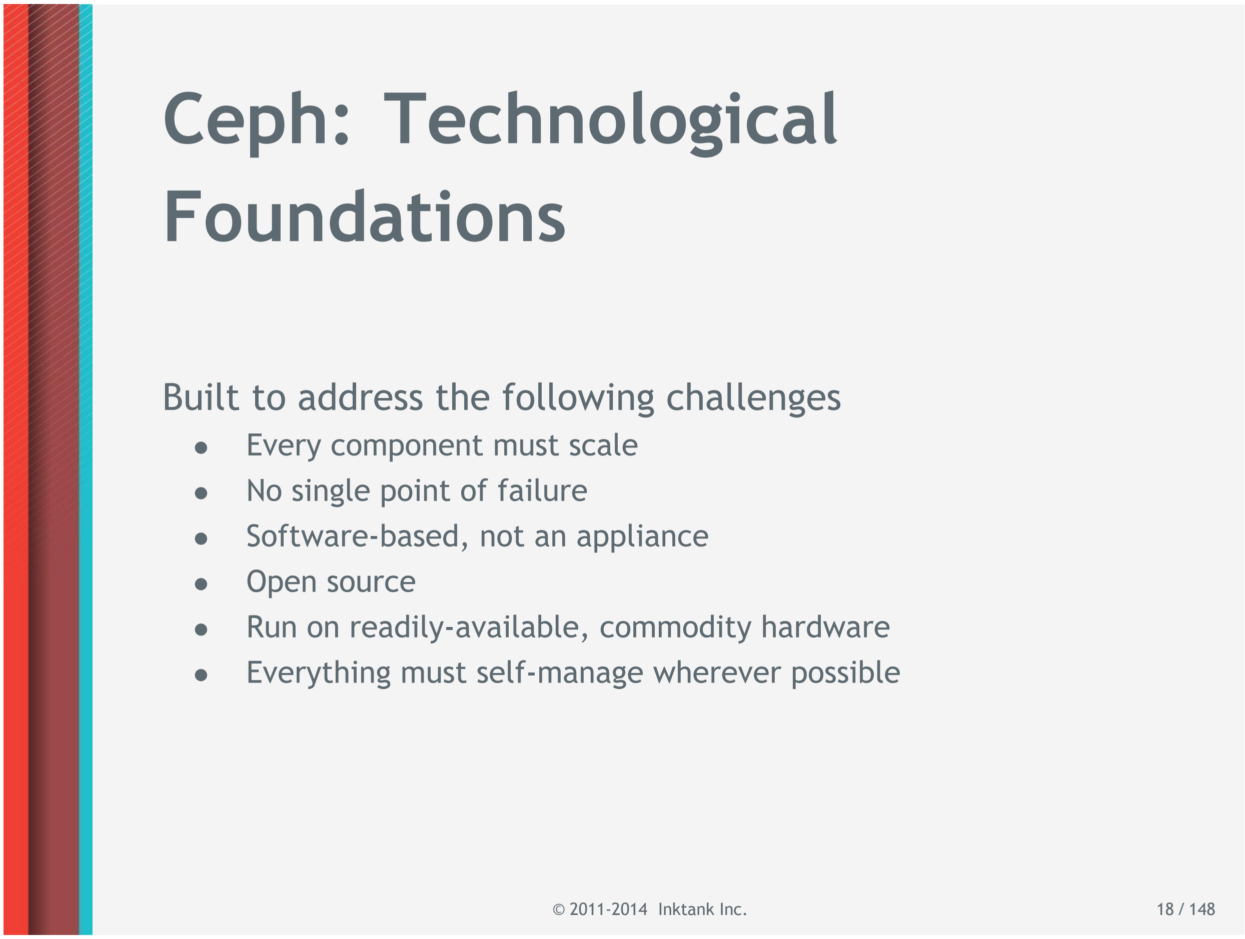
CEPH STORAGE CLUSTER

RADOS

A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

17 / 148

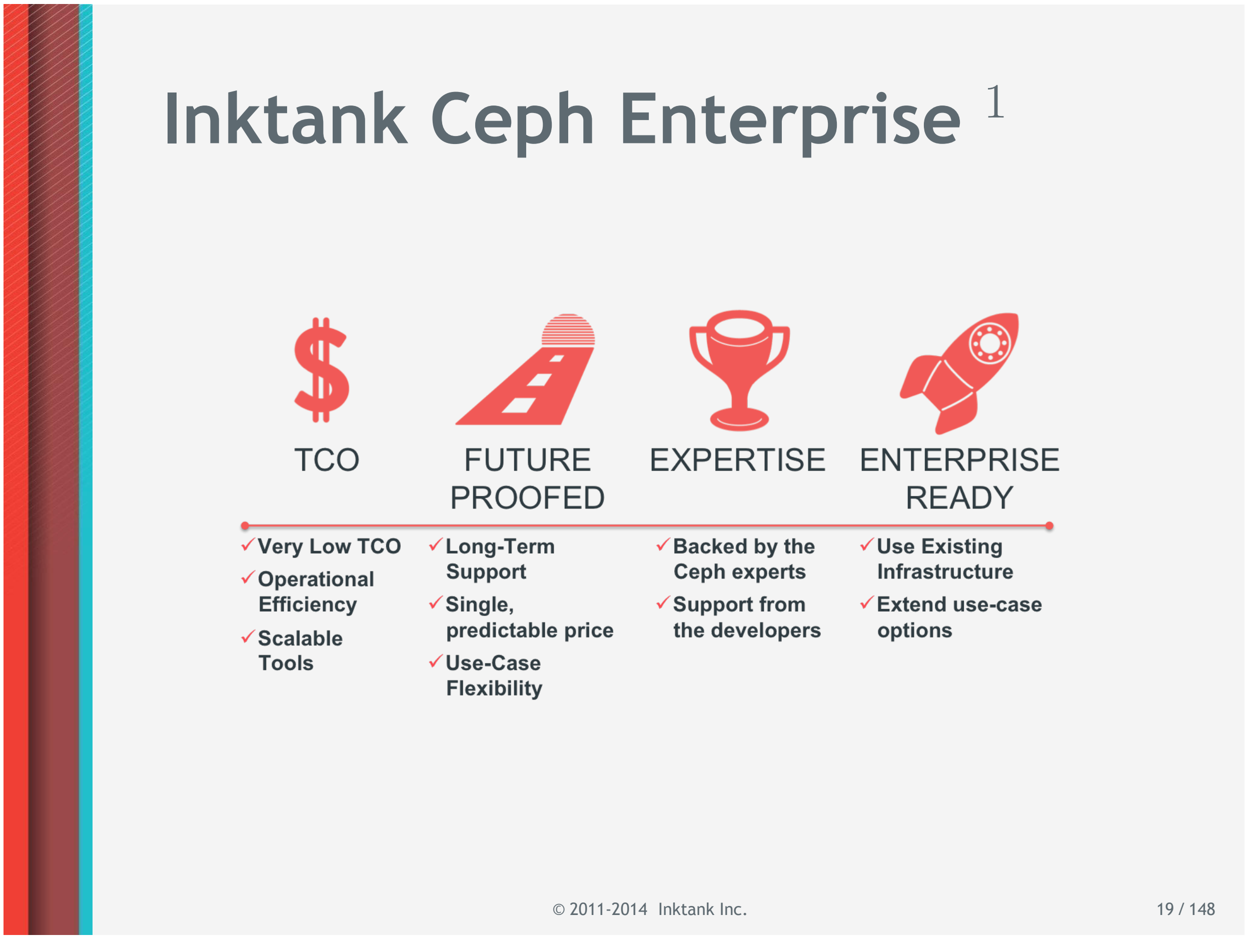
All the analysts will tell you that we□re facing a data explosion. If you are responsible for managing data for your company, you don□t need the analysts to tell you that. As disks become less expensive, there are easier for users to generate content. And that content must be managed, protected, and backed up so that it is available to you users whenever they request it.



Ceph: Technological Foundations

Built to address the following challenges

- Every component must scale
- No single point of failure
- Software-based, not an appliance
- Open source
- Run on readily-available, commodity hardware
- Everything must self-manage wherever possible



Inktank Ceph Enterprise [1]

TCO

- ✓Very Low TCO
- ✓Operational Efficiency
- ✓Scalable Tools

FUTURE PROOFED

- ✓Long-Term Support
- ✓Single, predictable price
- ✓Use-Case Flexibility

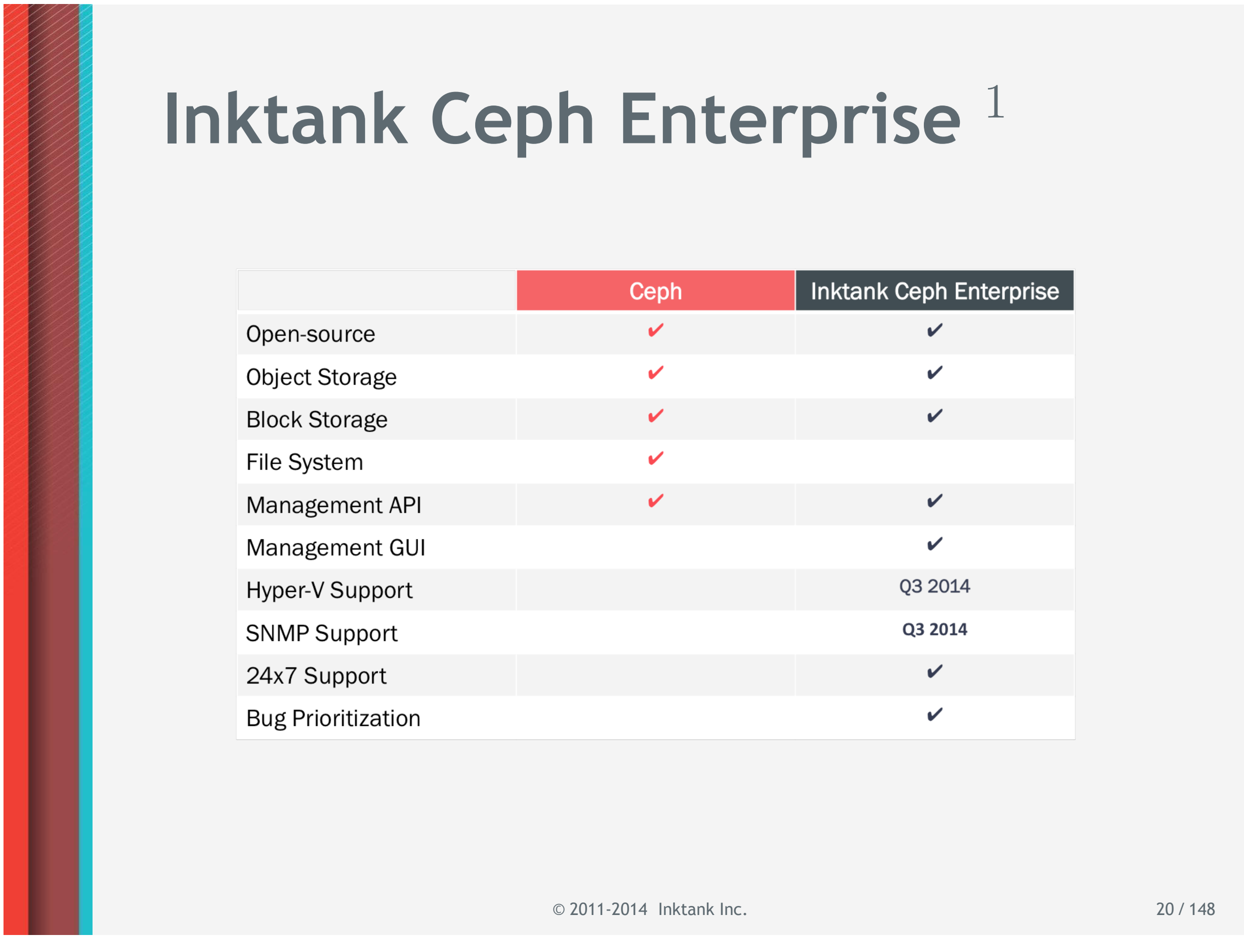
EXPERTISE

- ✓Backed by the Ceph experts
- ✓Support from the developers

ENTERPRISE READY

- ✓Use Existing Infrastructure
- ✓Extend use-case options

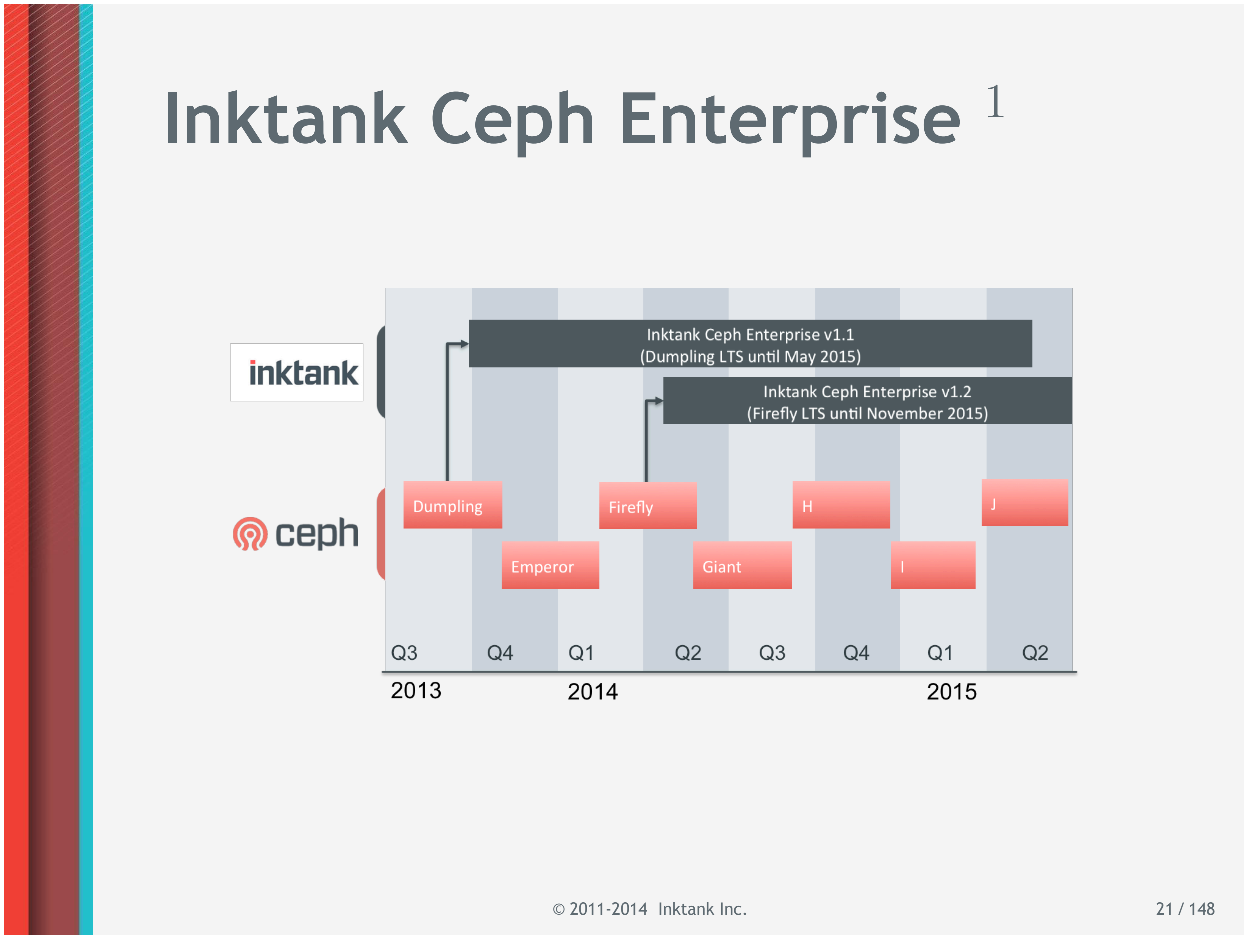
Note 1 : Inktank Ceph Enterprise



Inktank Ceph Enterprise [1]

	Ceph	Inktank Ceph Enterprise
Open-source	✓	✓
Object Storage	✓	✓
Block Storage	✓	✓
File System	✓	
Management API	✓	✓
Management GUI		✓
Hyper-V Support		Q3 2014
SNMP Support		**Q3 2014**
24x7 Support		✓
Bug Prioritization		✓

Note 1 : Inktank Ceph Enterprise



Inktank Ceph Enterprise [1]

inktank

ceph

Inktank Ceph Enterprise v1.1
(Dumpling LTS until May 2015)

Inktank Ceph Enterprise v1.2
(Firefly LTS until November 2015)

Dumpling

Emperor

Firefly

Giant

H

I

J

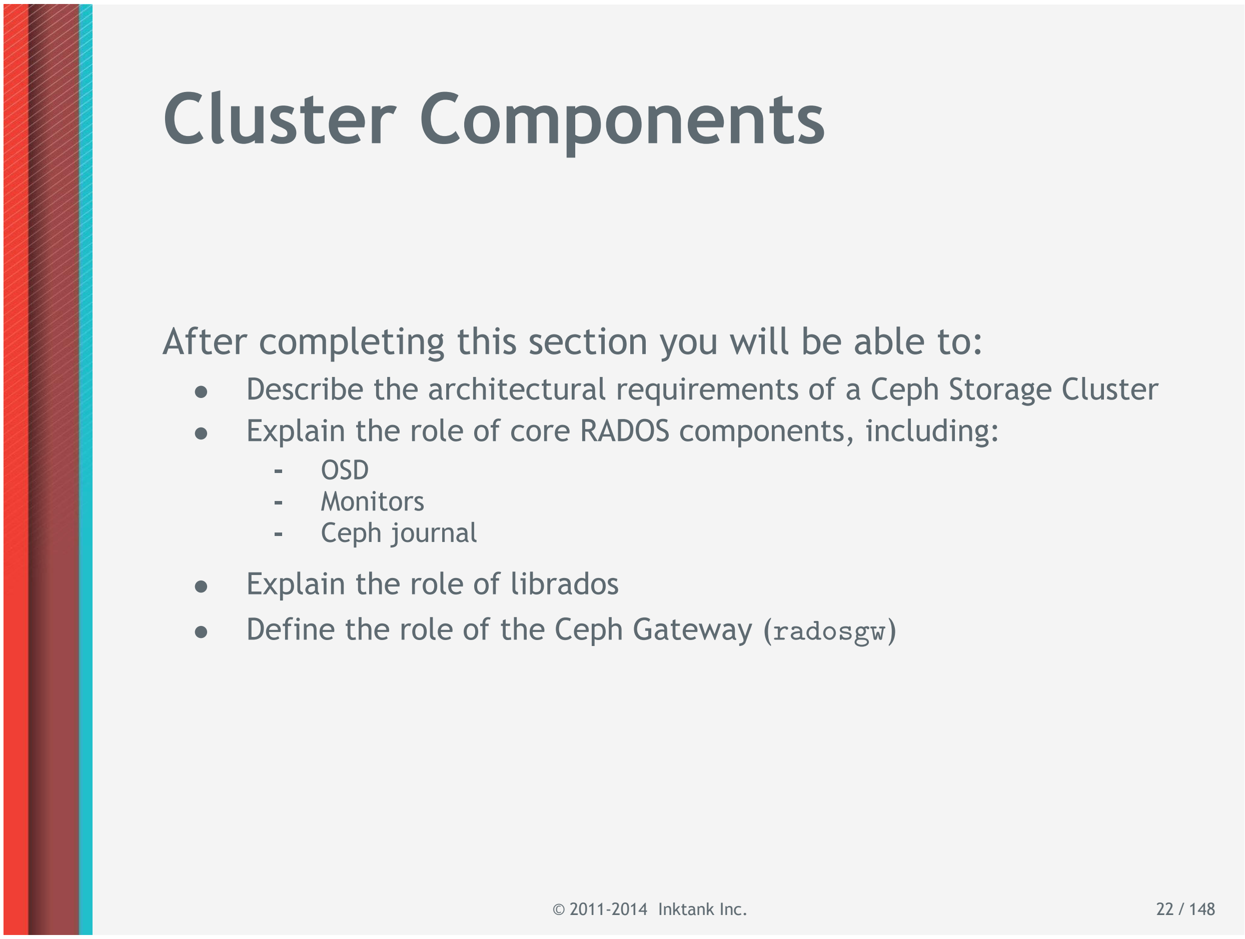
Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2

2013 2014 2015

© 2011-2014 Inktank Inc.

21 / 148

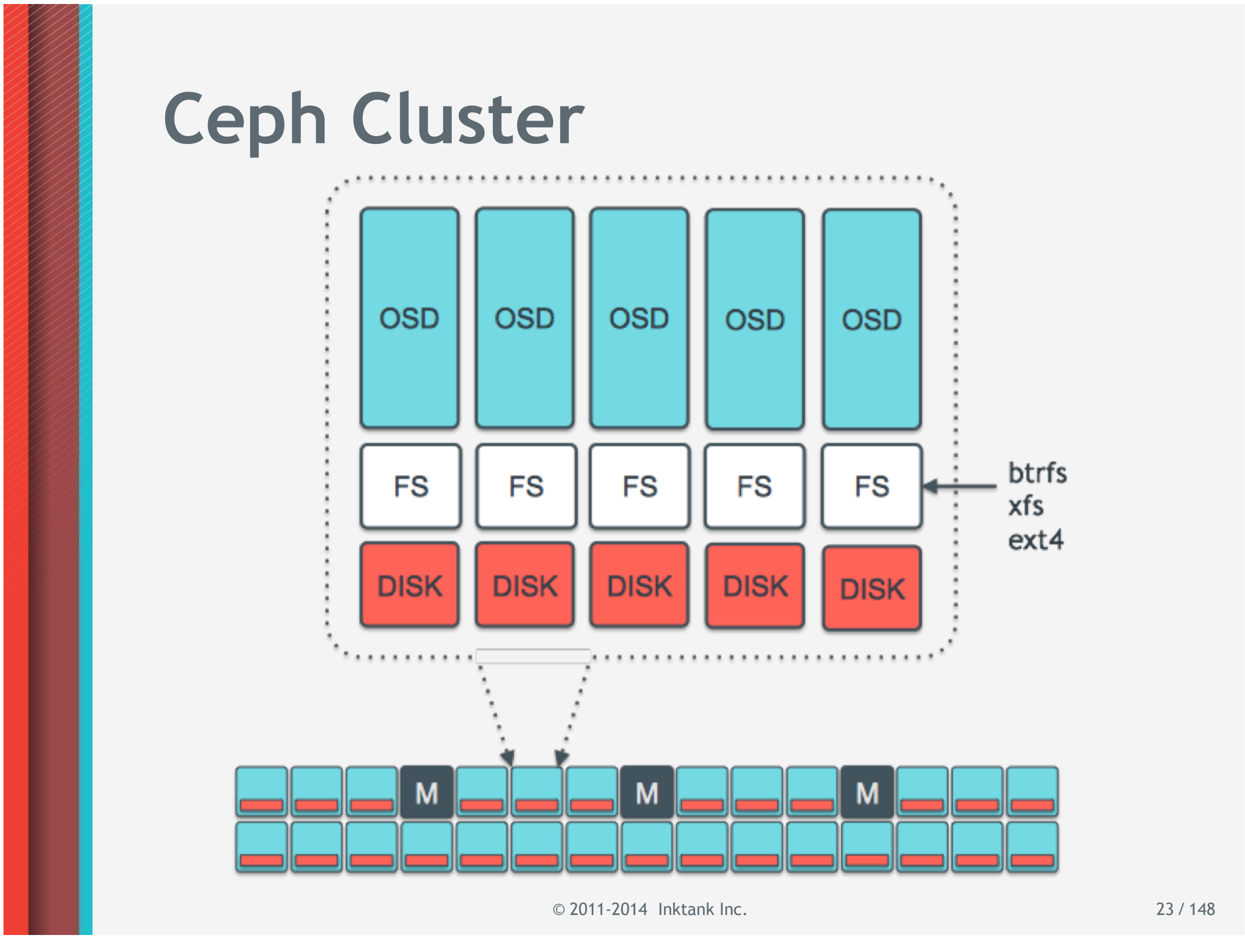
[1] Note 1 : Inktank Ceph Enterprise



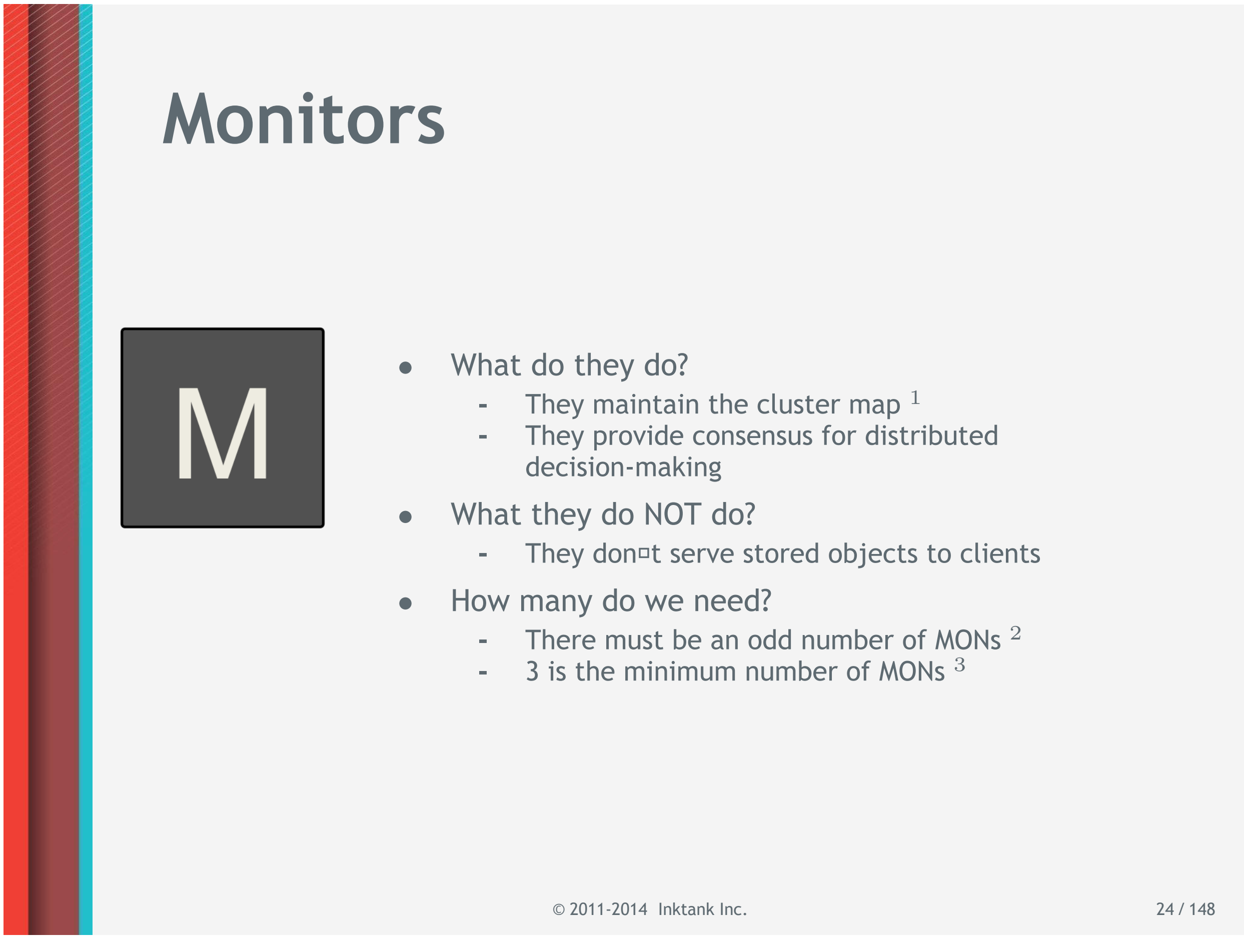
Cluster Components

After completing this section you will be able to:

- Describe the architectural requirements of a Ceph Storage Cluster
- Explain the role of core RADOS components, including:
  - OSD
  - Monitors
  - Ceph journal
- Explain the role of librados
- Define the role of the Ceph Gateway (`radosgw`)



Ceph Cluster



Monitors

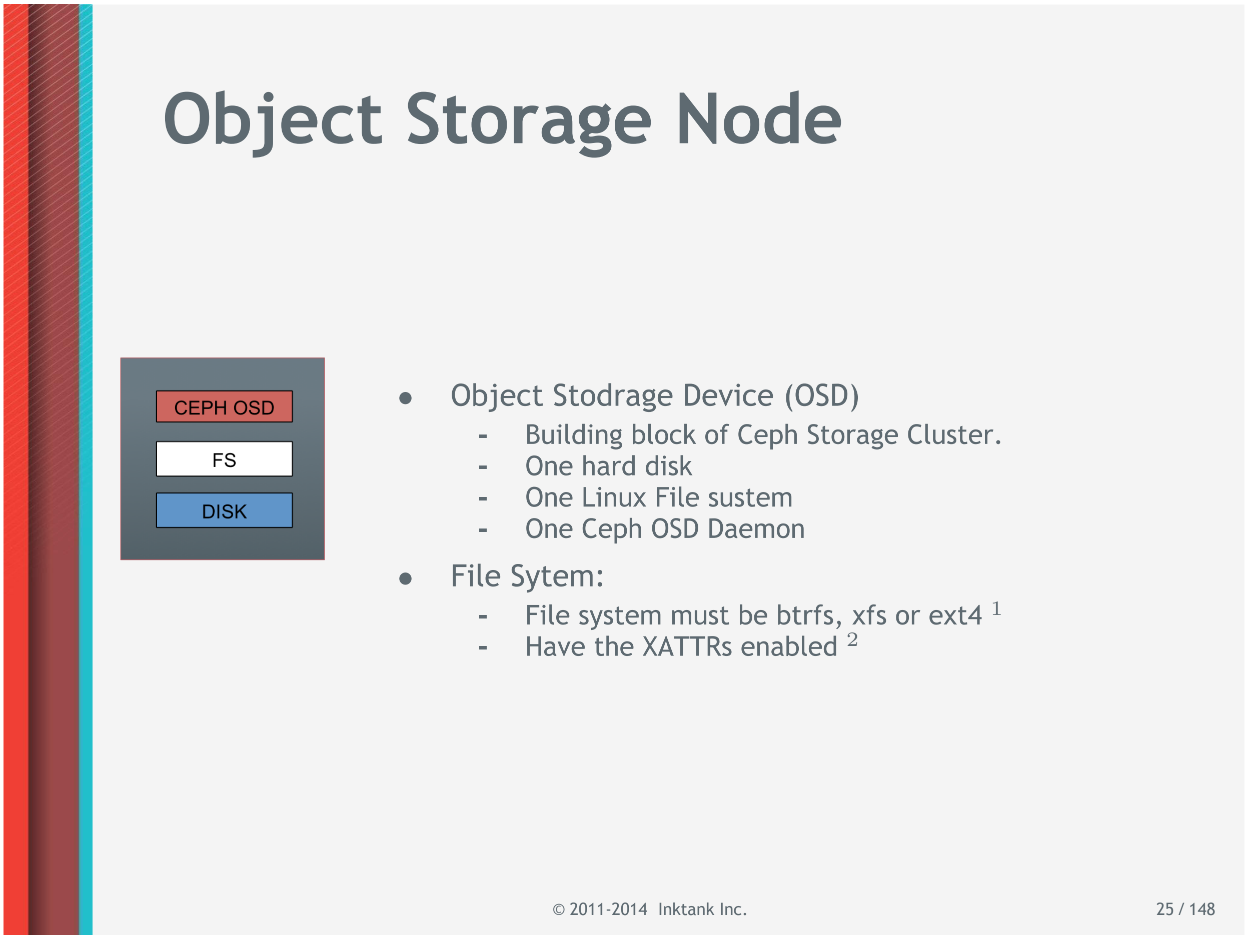
- What do they do?
  - They maintain the cluster map [1]
  - They provide consensus for distributed decision-making
- What they do NOT do?
  - They don□t serve stored objects to clients
- How many do we need?
  - There must be an odd number of MONs [2]
  - 3 is the minimum number of MONs [3]

Note 1: Ceph Monitors are daemons. The primary role of a monitor is to maintain the state of the cluster by managing critical Ceph Cluster state and configuration information. The Ceph Monitors maintain a master copy of the CRUSH Map and Ceph Daemons and Clients can check in periodically with the monitors to be sure they have the most recent copy of the map.

Note 2: The monitors most establish a consensus regarding the state of the cluster, which is why there must be an odd number of monitors.

Note 3: In critical environment and to provide even more reliability and fault tolerance, it can be advised to run up 5 Monitors

In order for the Ceph Storage Cluster to be operational and accessible, there must be at least more than half of Monitors running and operational. If this number goes below, and as Ceph will always guarantee the integrity of the data to its accessibility, the complete Ceph Storage Cluster will become inaccessible to any client.
For your information, the Ceph Storage Cluster maintains different map for its operations:
- MON Map
- OSD Map
- CRUSH Map


CEPH OSD
FS
DISK

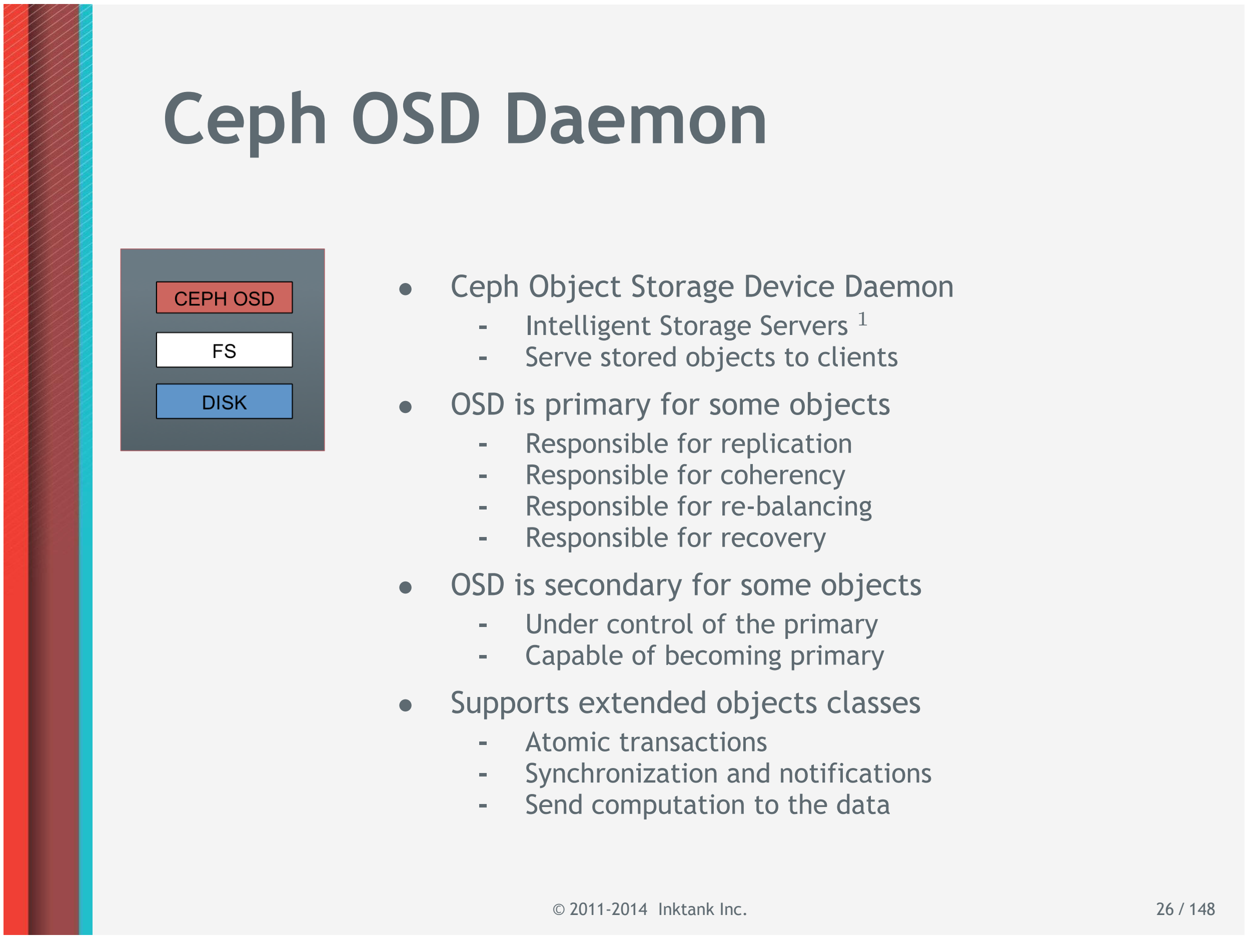
Object Storage Node

- Object Stodrage Device (OSD)
  - Building block of Ceph Storage Cluster.
  - One hard disk
  - One Linux File sustem
  - One Ceph OSD Daemon
- File Sytem:
  - File system must be btrfs, xfs or ext4 [1]
  - Have the XATTRs enabled [2]

The OSD connect a disk to the Ceph Storage Cluster

Note 1: The only system supported in the dumpling and emperor versions are XFS and EXT4. BTRFS, although very promising, is not yet ready for production environments. There is also work ongoing in order to someday provide support around ZFS

Note 2: The Extended Attributes of the underlying file system are used for storing and retrieving information about the internal object state, snapshot metadata, and Ceph Gateway access control lists (ACLs)

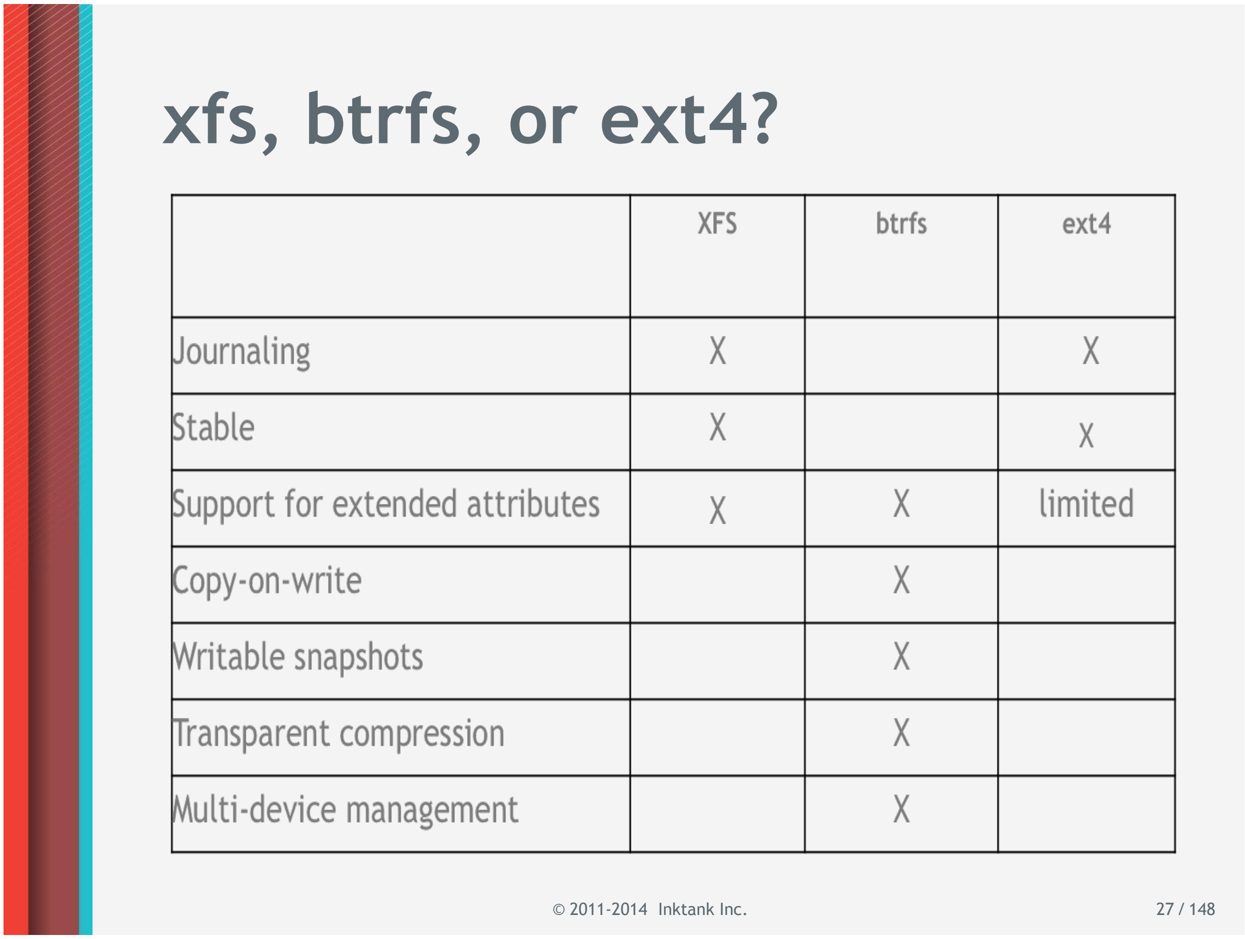

CEPH OSD
FS
DISK

Ceph OSD Daemon

- Ceph Object Storage Device Daemon
  - Intelligent Storage Servers [1]
  - Serve stored objects to clients
- OSD is primary for some objects
  - Responsible for replication
  - Responsible for coherency
  - Responsible for re-balancing
  - Responsible for recovery
- OSD is secondary for some objects
  - Under control of the primary
  - Capable of becoming primary
- Supports extended objects classes
  - Atomic transactions
  - Synchronization and notifications
  - Send computation to the data

Note 1: The overall design and goal of the OSD is to bring the computing power as close as possible of the data and to let it perform the maximum it can. For now, it processes the functions listed in the bullet lists, depending on its role (primary or secondary), but in the future, Ceph will probably leverage the close link between the OSD and the data to extend the computational power of the OSD.

For example: The OSD drive creation of the thumbnail of an object rather than having the client being responsible for such an operation.

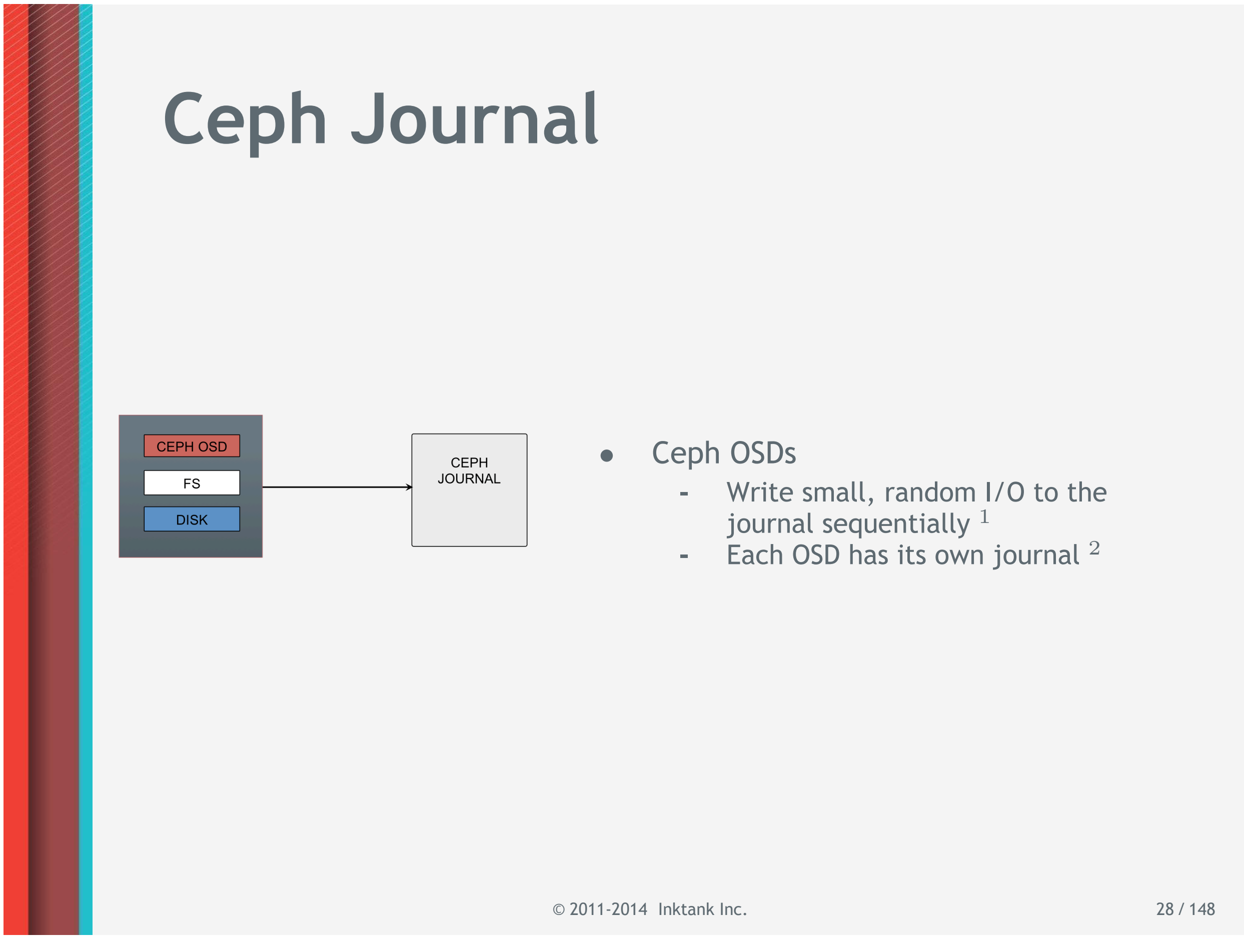


xfs, btrfs, or ext4?

	XFS	btrfs	ext4
Journaling	X		X
Stable	X		X
Support for extended attributes	X	X	limited
Copy-on-write		X	
Writable snapshots		X	
Transparent compression		X	
Multi-device management		X	

Ceph requires a modern Linux file system. We have tested XFS, btrs and ext4, and these are the supported file systems. Full size and extensive tests have been performed on BTRFS is not recommended for productions environments.

Right now for stability, the recommendation os to use xfs.



Ceph Journal

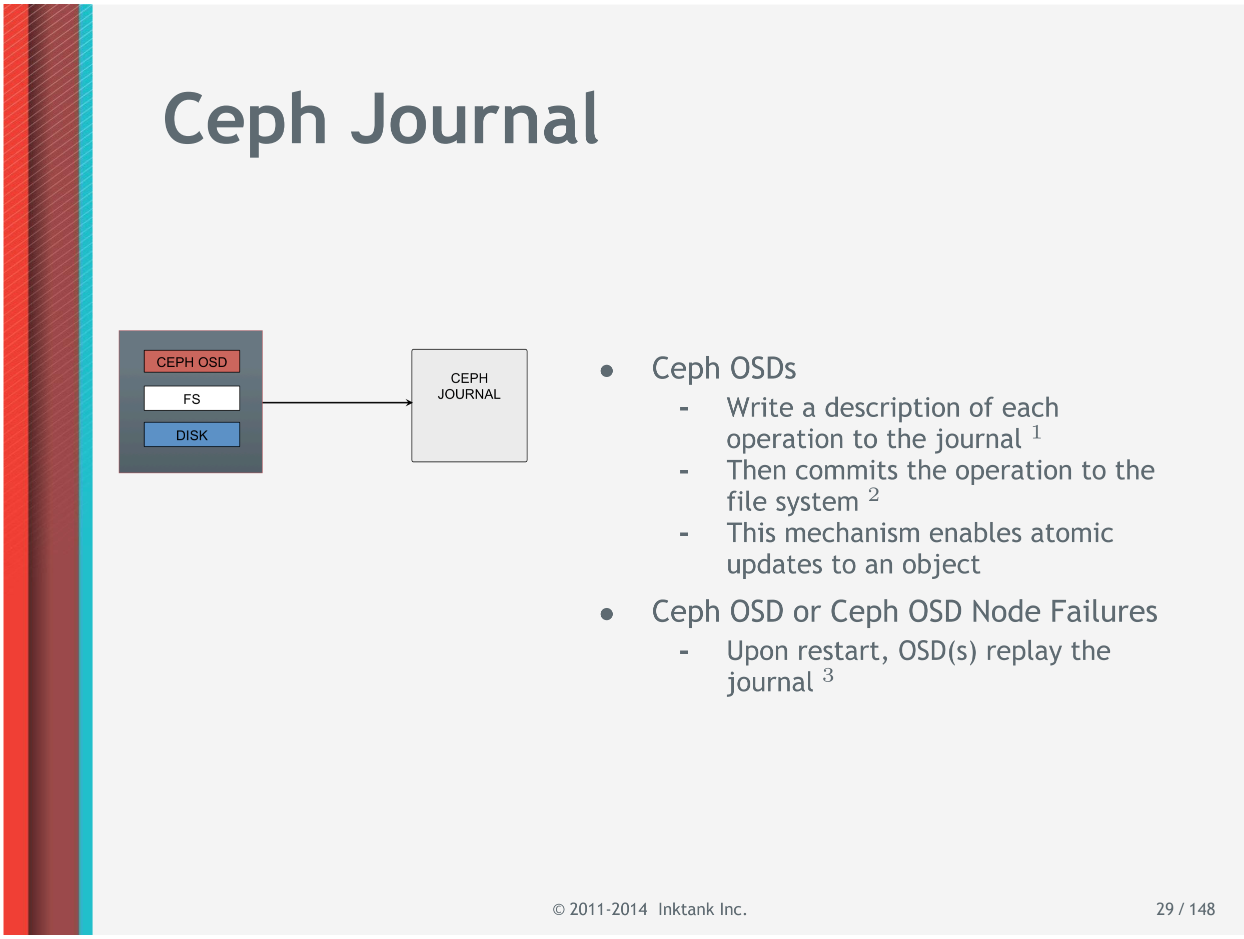
CEPH OSD
FS
DISK
CEPH JOURNAL

- Ceph OSDs
  - Write small, random I/O to the journal sequentially [1]
  - Each OSD has its own journal [2]

Journals use raw volumes on the OSD nodes

Note 1 : This is done for speed and consistency. It speeds up workloads by allowing the host file system more time to coalesce writes because of the small IO request being performed

Note 2 : By default, the Ceph Journal is written to the same disk as the OSD data. For better performance, the Ceph Journal should be configured to write to its own hard drive such as a SSD.



Ceph Journal

CEPH OSD

FS

DISK

CEPH JOURNAL

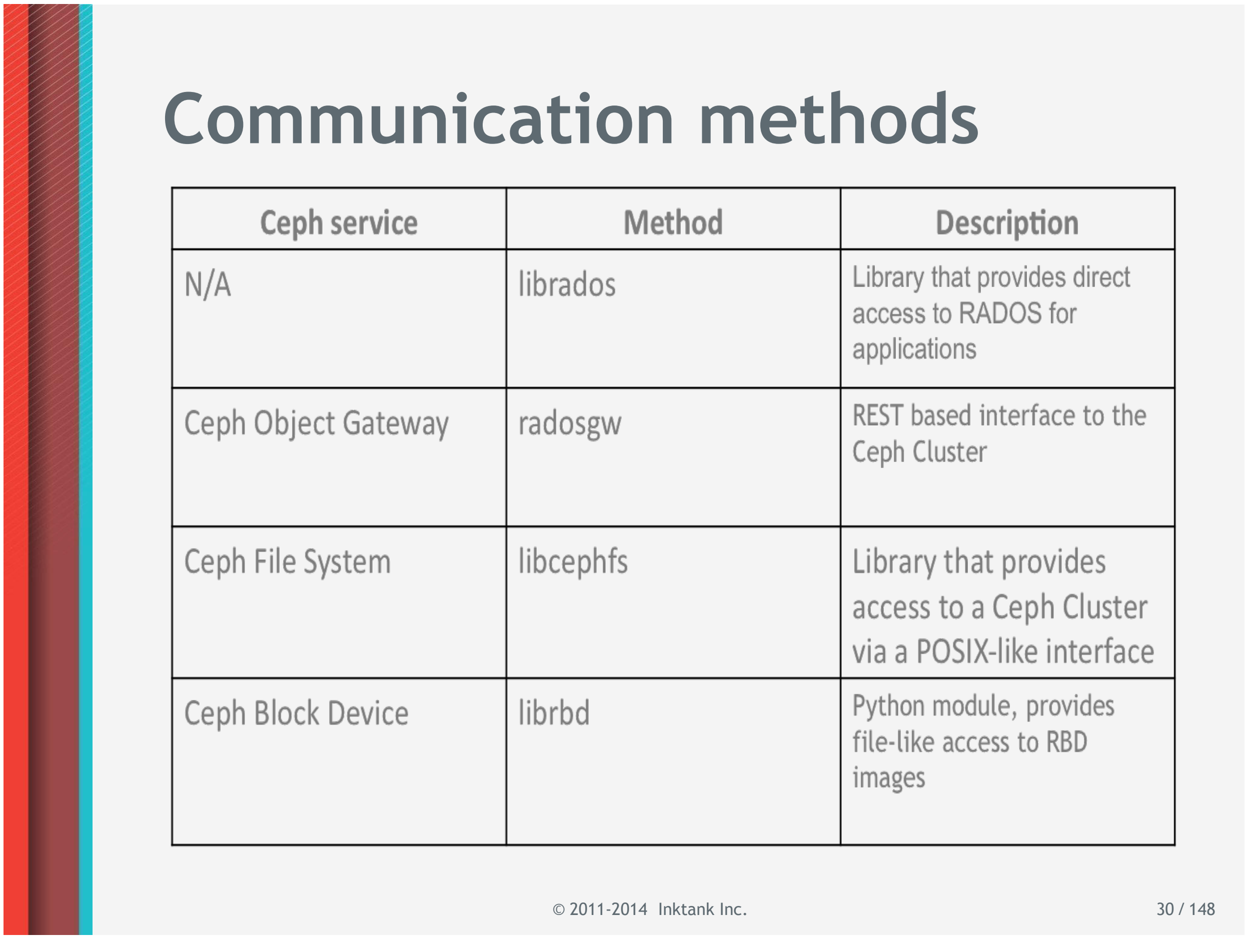
- Ceph OSDs
  - Write a description of each operation to the journal [1]
  - Then commits the operation to the file system [2]
  - This mechanism enables atomic updates to an object
- Ceph OSD or Ceph OSD Node Failures
  - Upon restart, OSD(s) replay the journal [3]

29 / 148

Note 1 : The write to the CEPH cluster will be acknowledged when the minimum number of replica journals have been written to.

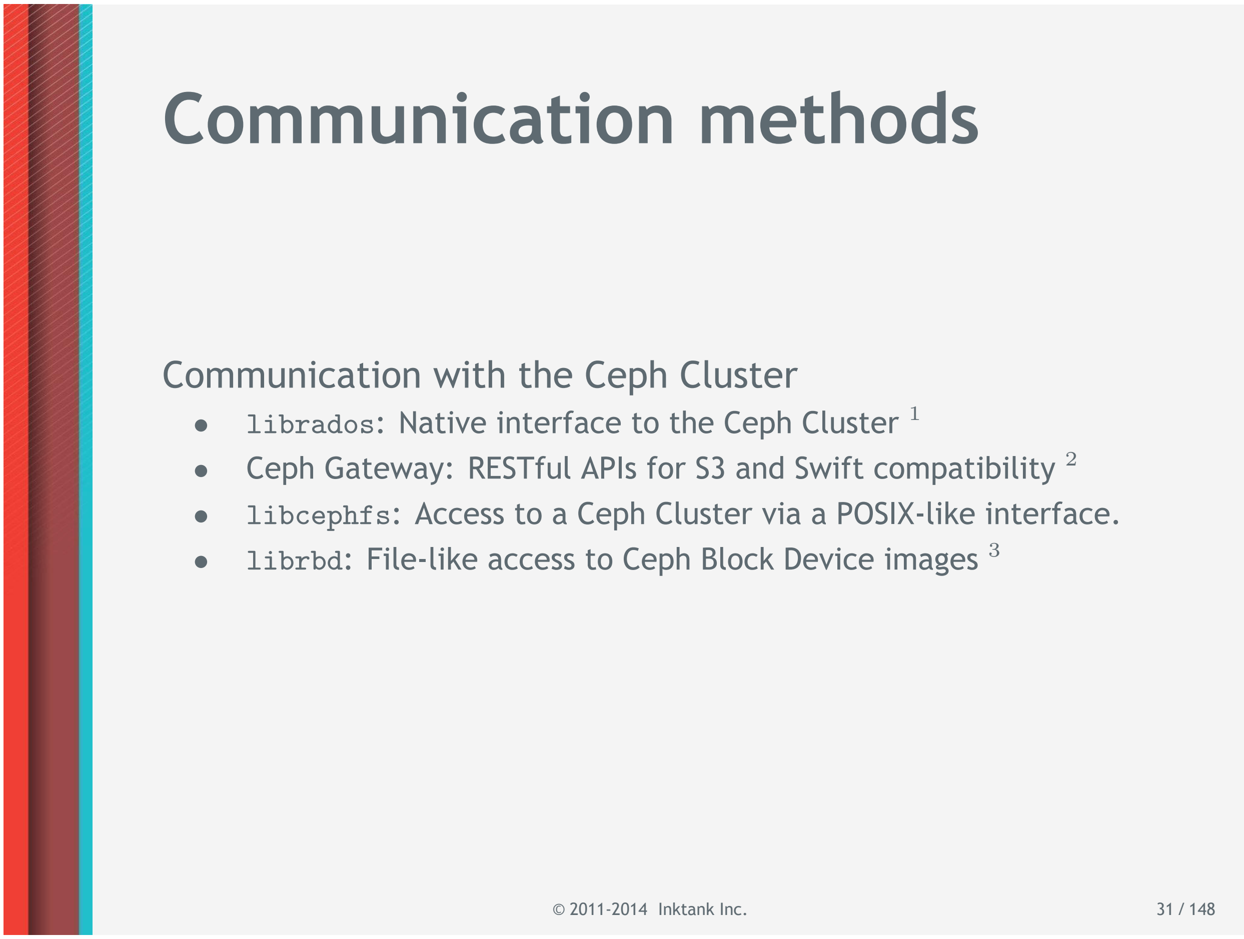
Note 2 : The OSD stops writing every few seconds and synchronizes the journal with the file system commits performed so they can trim operations from the journal to reclaim the space on the journal disk volume.

Note 3 : The replay sequence will start after the last sync operation as previous journal records were trimmed out.



Communication methods

Ceph service	Method	Description
N/A	librados	Library that provides direct access to RADOS for applications
Ceph Object Gateway	radosgw	REST based interface to the Ceph Cluster
Ceph File System	libcephfs	Library that provides access to a Ceph Cluster via a POSIX-like interface
Ceph Block Device	librbd	Python module, provides file-like access to RBD images



Communication methods

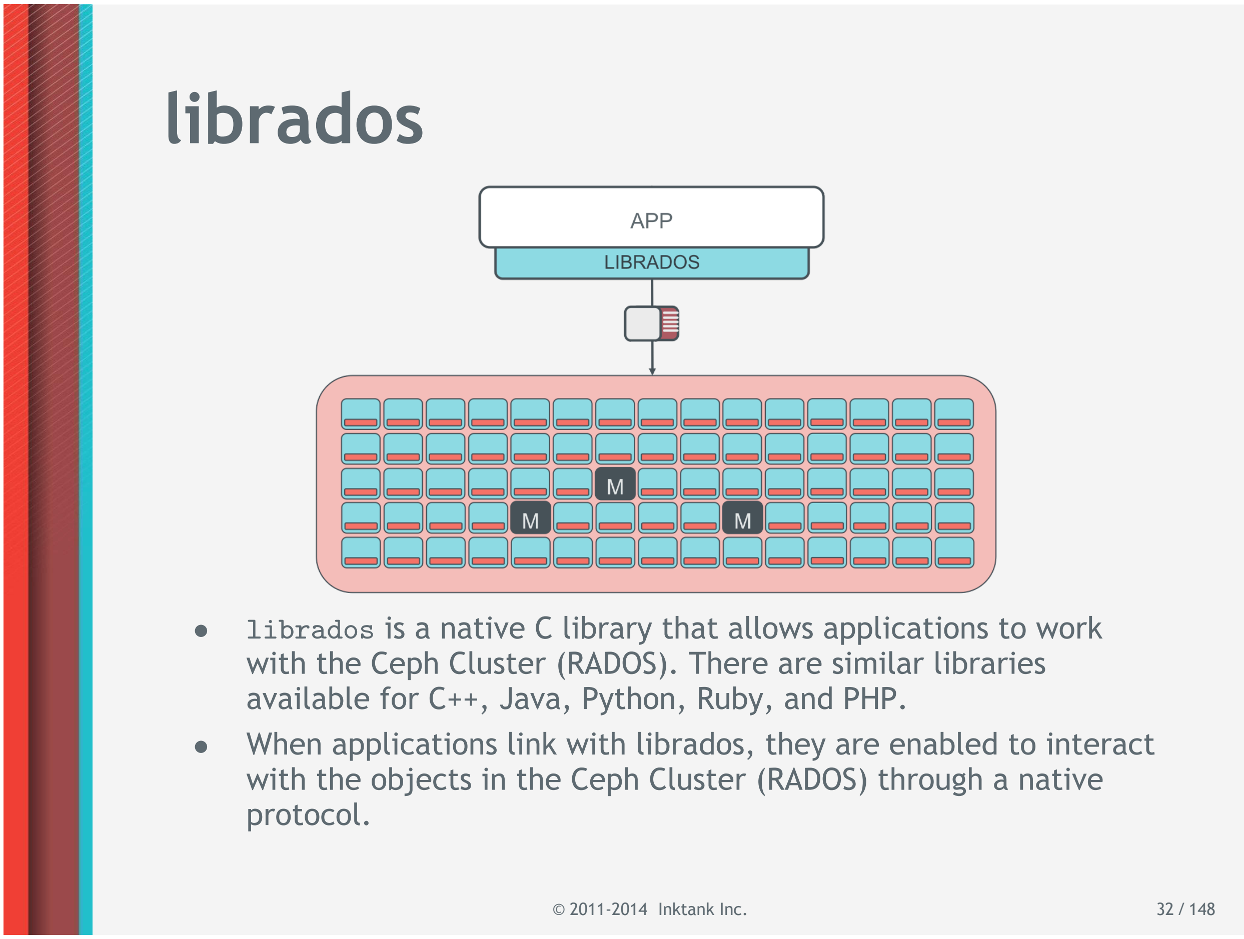
Communication with the Ceph Cluster

- `librados`: Native interface to the Ceph Cluster [1]
- Ceph Gateway: RESTful APIs for S3 and Swift compatibility [2]
- `libcephfs`: Access to a Ceph Cluster via a POSIX-like interface.
- `librbd`: File-like access to Ceph Block Device images [3]

Note 1: Services interfaces built on top of this native interface include the Ceph Block Device, The Ceph Gateway, and the Ceph File System.

Note 2: Amazon S3 and OpenStack Swift. The Ceph Gateway is referred to as `radosgw`.

Note 3: Python module

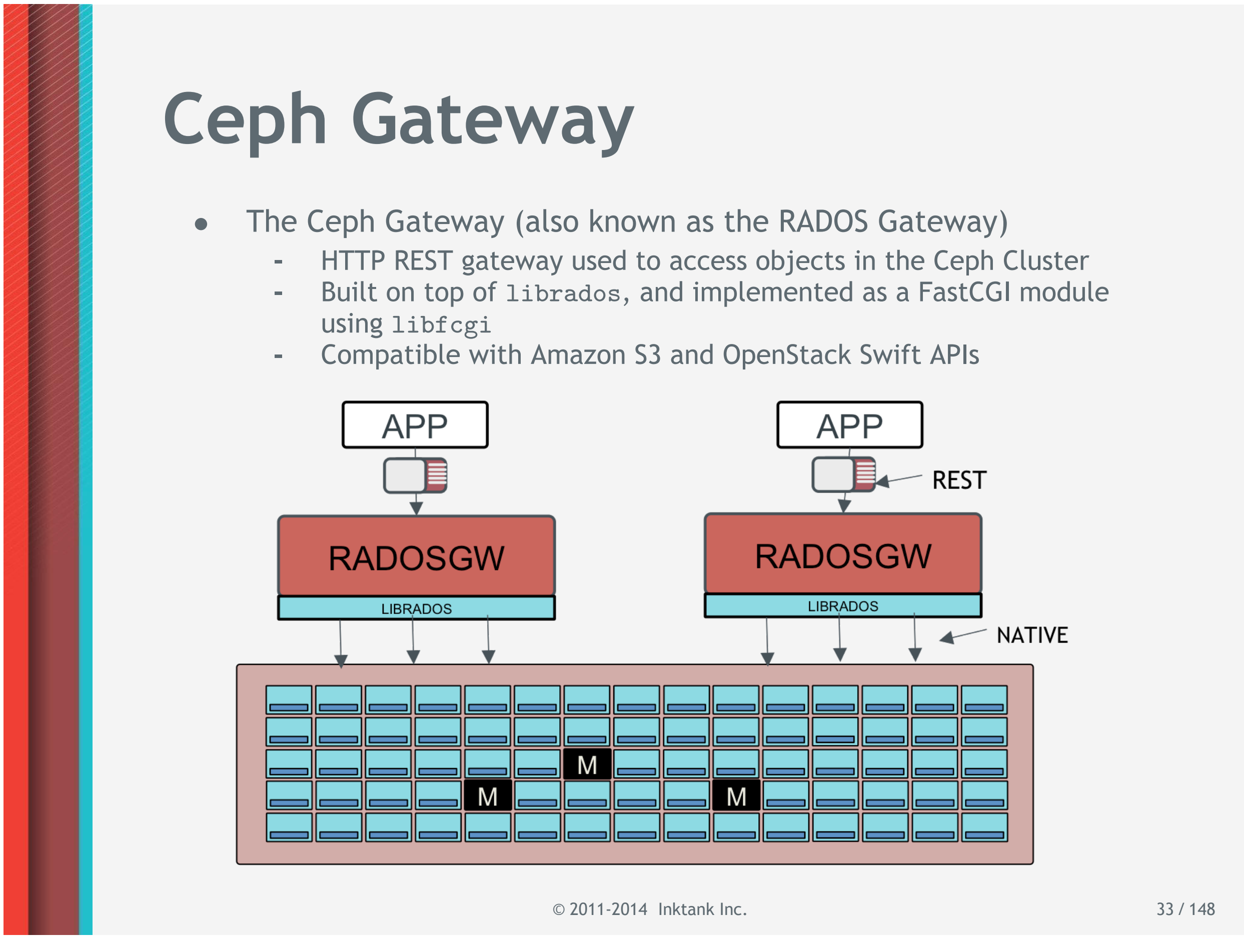


librados

- `librados` is a native C library that allows applications to work with the Ceph Cluster (RADOS). There are similar libraries available for C++, Java, Python, Ruby, and PHP.
- When applications link with librados, they are enabled to interact with the objects in the Ceph Cluster (RADOS) through a native protocol.

`librados` is a native C library that allows applications to work with the Ceph Cluster (RADOS). There are similar libraries available for C++, Java, Python, Ruby, and PHP.

When applications link with `librados`, they are enabled to interact with the objects in the Ceph Cluster (RADOS) through a native protocol.



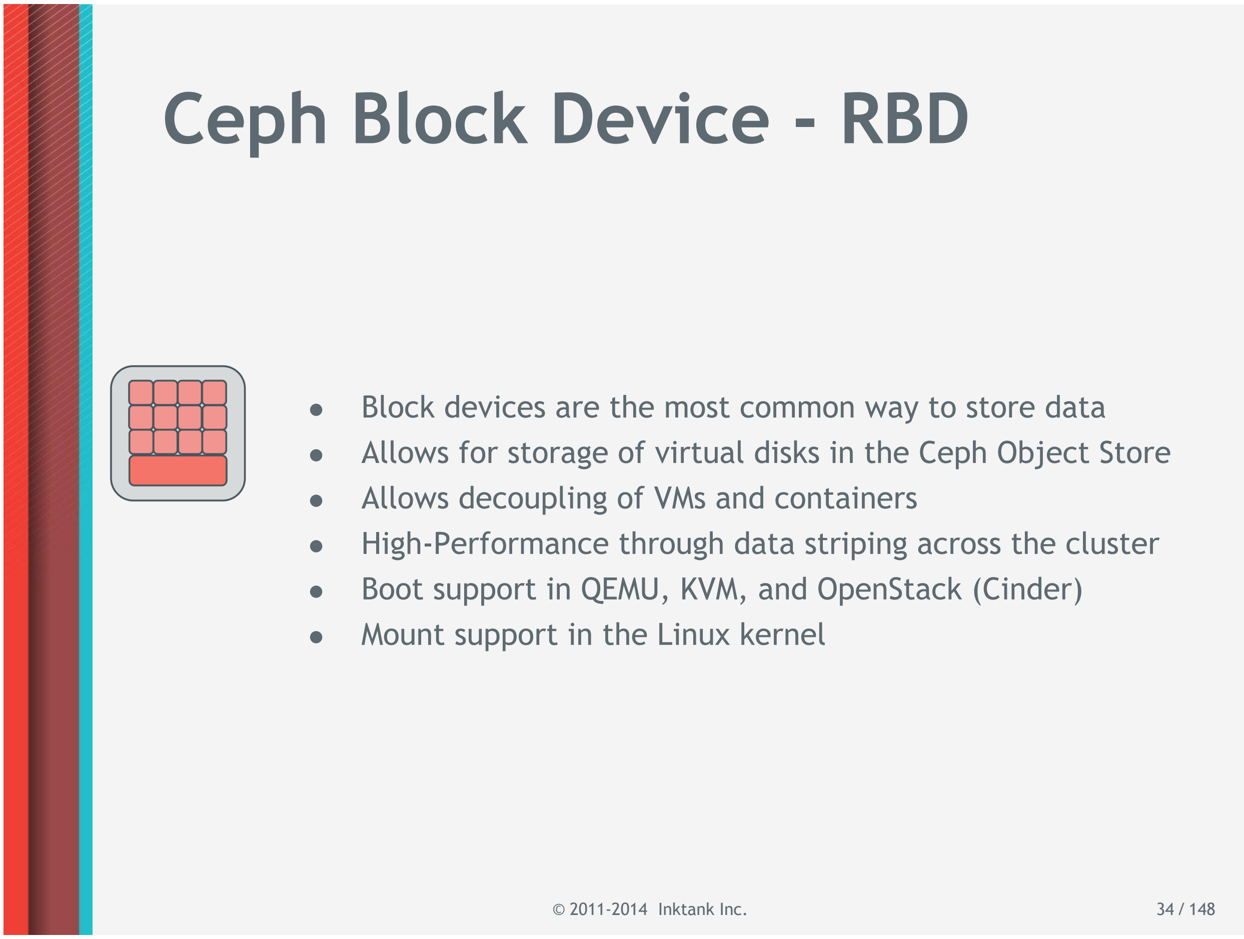
Ceph Gateway

- The Ceph Gateway (also known as the RADOS Gateway)
  - HTTP REST gateway used to access objects in the Ceph Cluster
  - Built on top of `librados`, and implemented as a FastCGI module using `libfcgi`
  - Compatible with Amazon S3 and OpenStack Swift APIs

© 2011-2014 Inktank Inc.

The gateway application sits on top of a webserver, it uses the librados library to communicate with the CEPH cluster and will write to OSD processes directly.

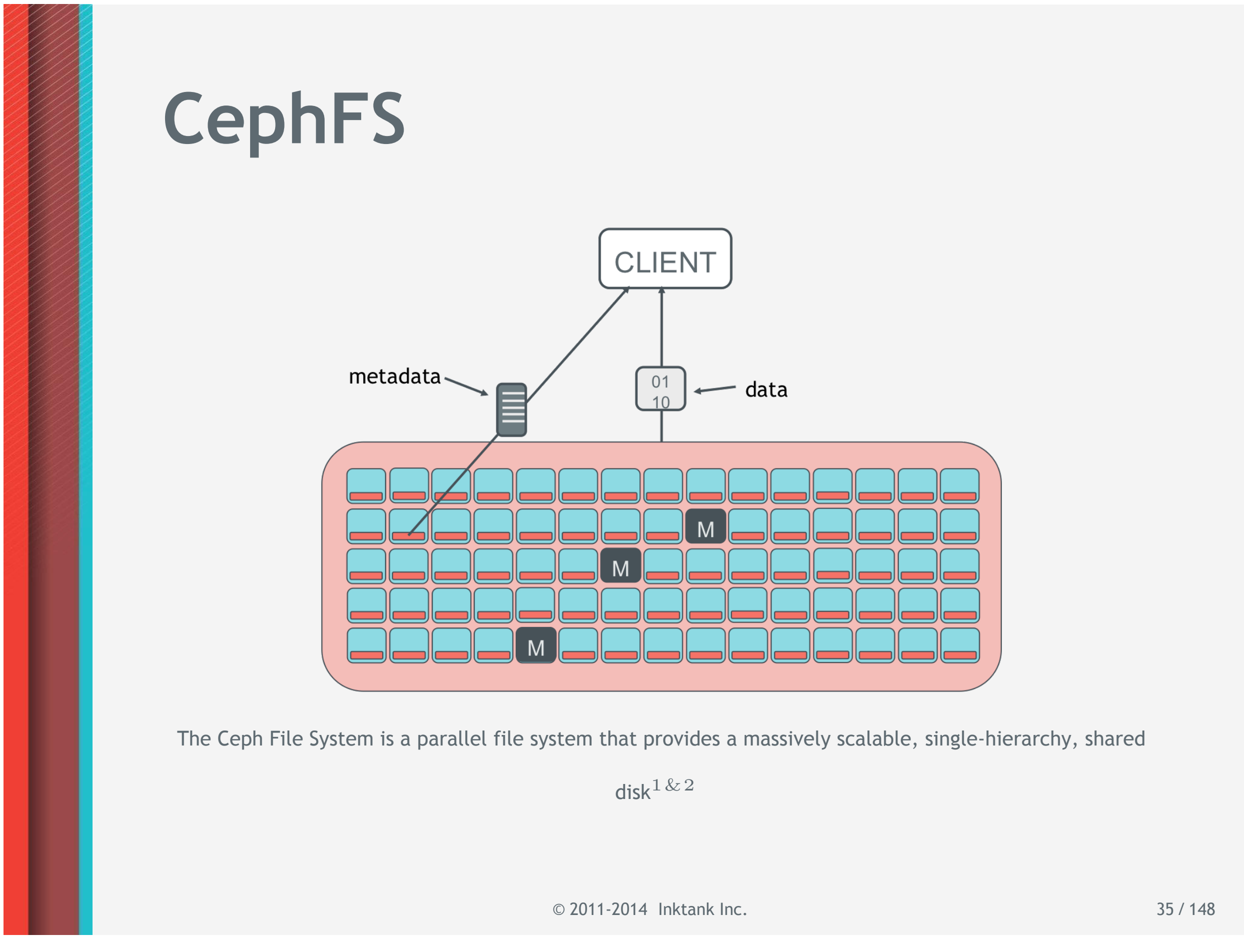
The Ceph Gateway (also known as the RADOS Gateway) is an HTTP REST gateway used to access objects in the Ceph Cluster. It is built on top of librados, and implemented as a FastCGI module using libfcgi, and can be used with any FastCGI capable web server. Because it uses a unified namespace, it is compatible with Amazon S3 RESTful APIs and OpenStack Swift APIs.



Ceph Block Device - RBD

- Block devices are the most common way to store data
- Allows for storage of virtual disks in the Ceph Object Store
- Allows decoupling of VMs and containers
- High-Performance through data striping across the cluster
- Boot support in QEMU, KVM, and OpenStack (Cinder)
- Mount support in the Linux kernel

RBD stands for RADOS Block Device
Date is striped across the Ceph cluster



CephFS

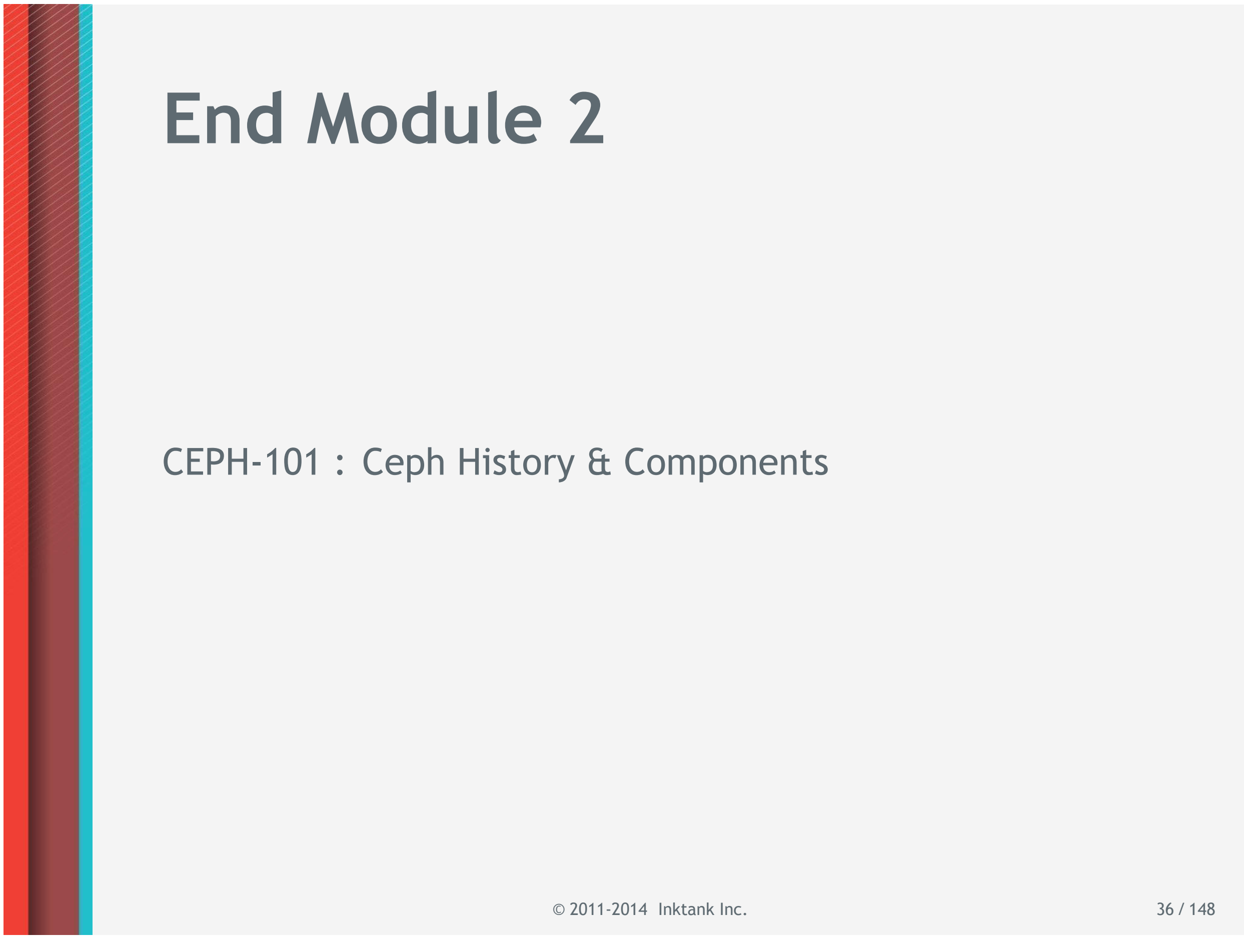
The Ceph File System is a parallel file system that provides a massively scalable, single-hierarchy, shared disk[1 & 2]

The MDS stores its data in the CEPH cluster.

It stores only metadata information for the files store in the file system (access, modify, create) dates for example.

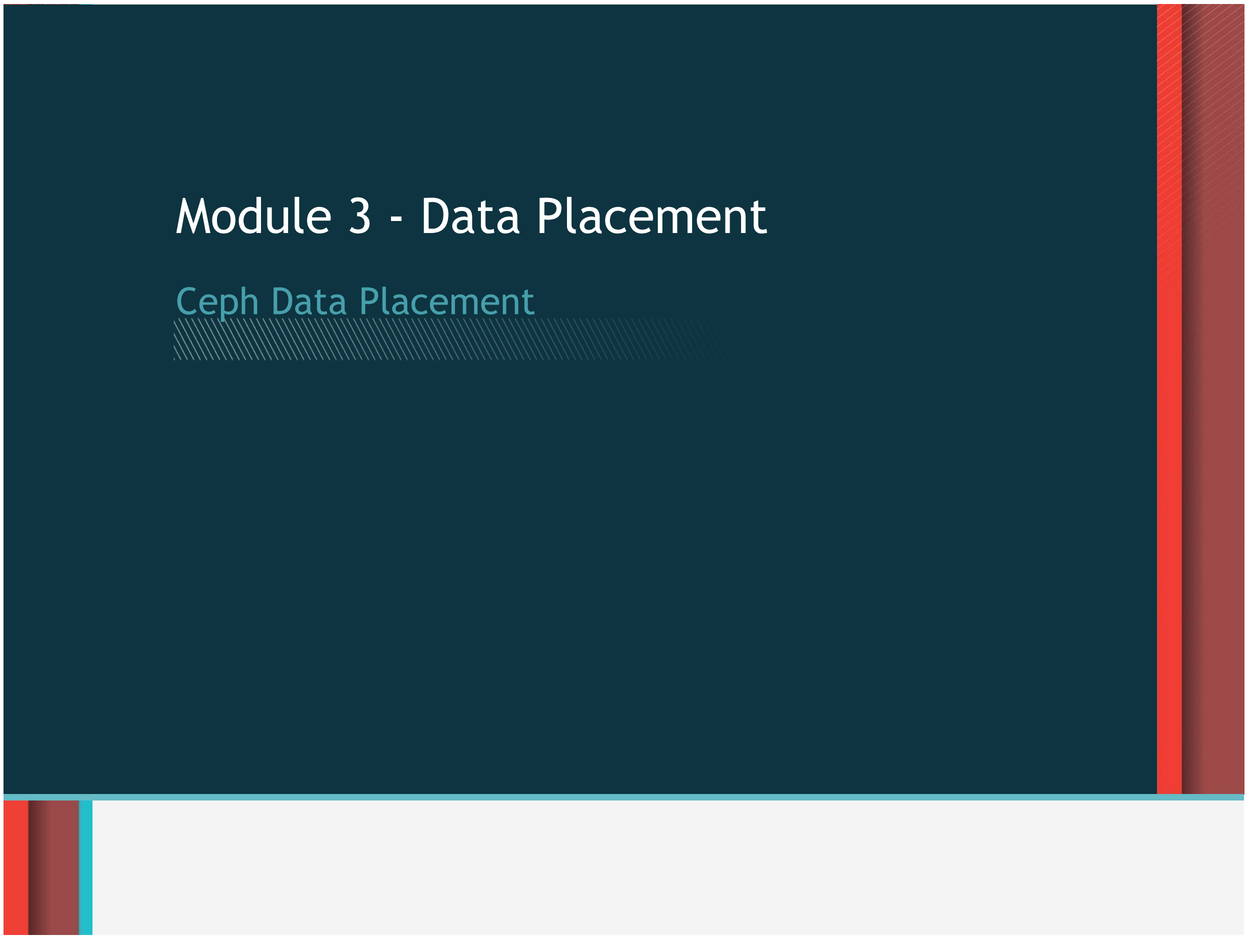
Note 1 : CephFS is not currently supported in production environments. It not in a production environment, you should only use an active/passive MDS configuration and not use snapshot.

Note 2 : CephFS should be not be mounted on a host that is a node in the Ceph Object Store



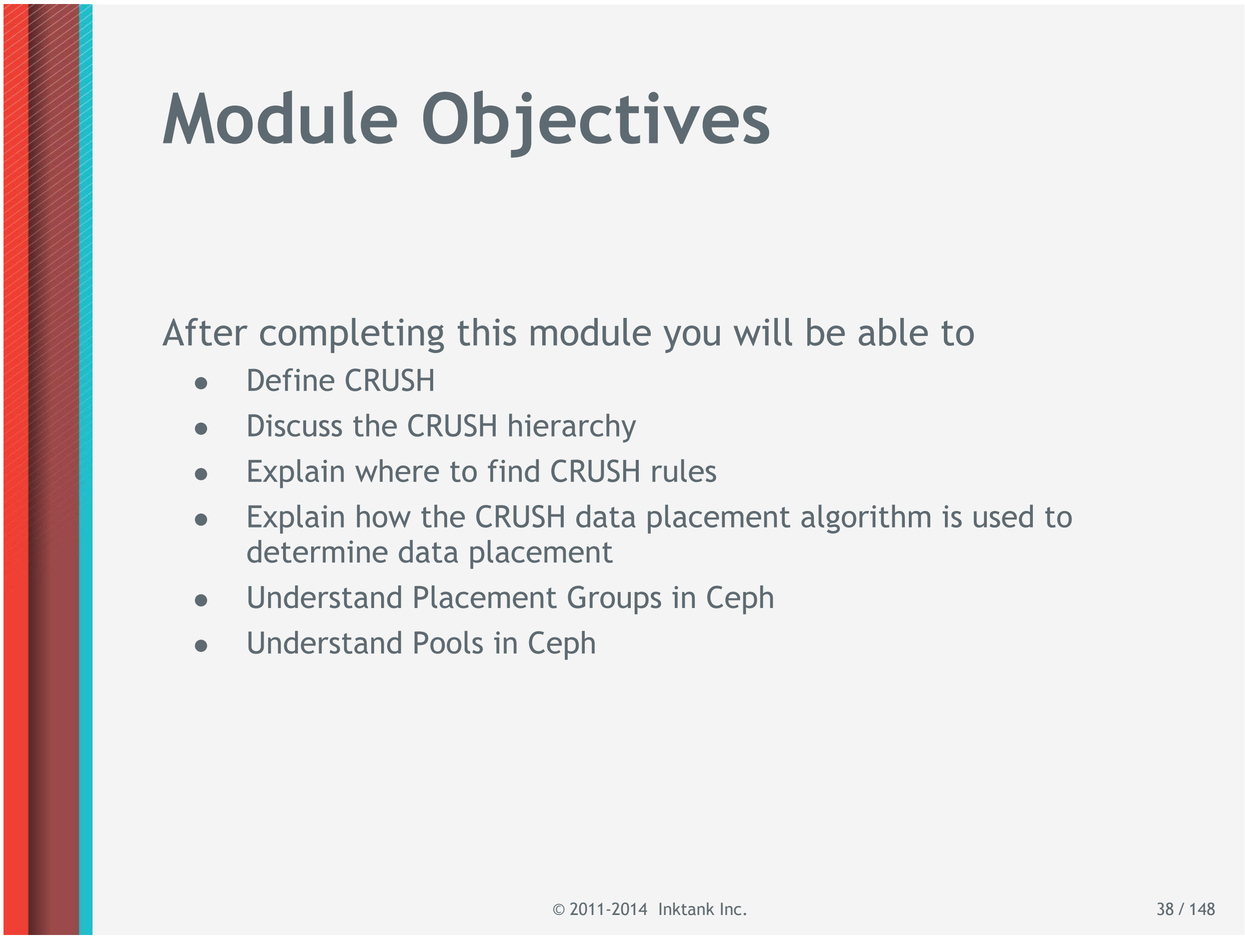
End Module 2

CEPH-101 : Ceph History & Components



Module 3 - Data Placement

Ceph Data Placement



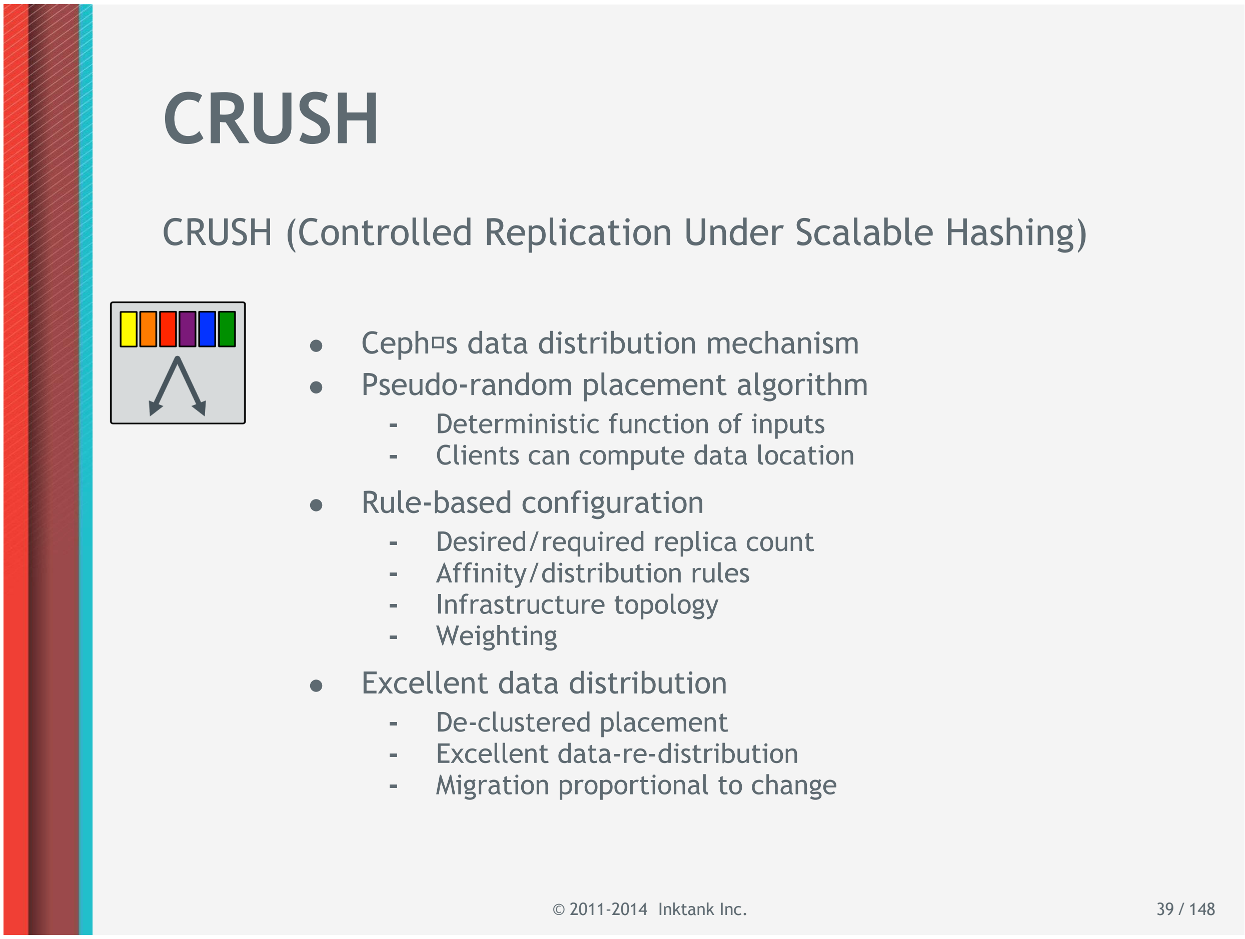
Module Objectives

After completing this module you will be able to

- Define CRUSH
- Discuss the CRUSH hierarchy
- Explain where to find CRUSH rules
- Explain how the CRUSH data placement algorithm is used to determine data placement
- Understand Placement Groups in Ceph
- Understand Pools in Ceph

What is CRUSH?

A pseudo random placement algorithm



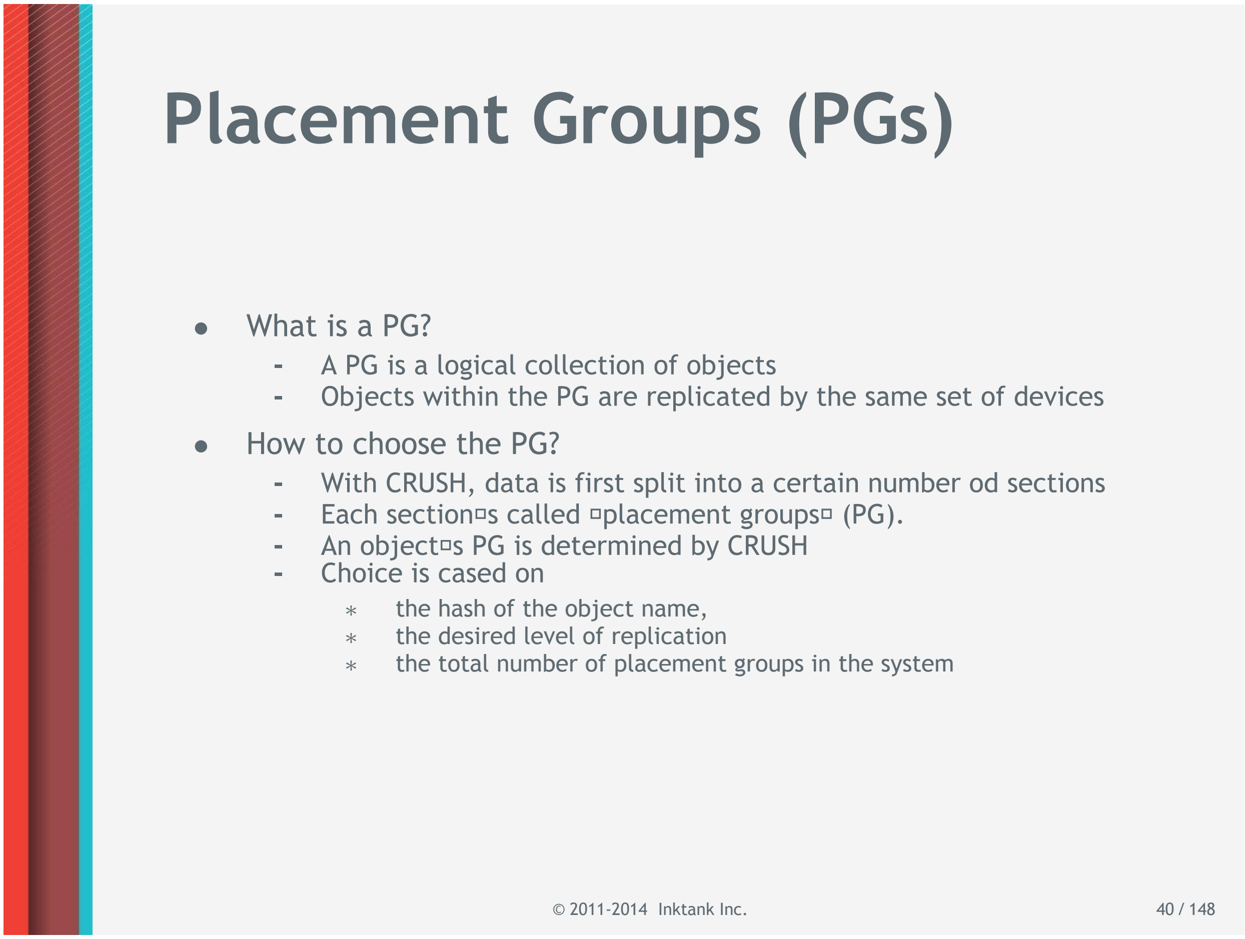
CRUSH

CRUSH (Controlled Replication Under Scalable Hashing)

- Ceph□s data distribution mechanism
- Pseudo-random placement algorithm
  - Deterministic function of inputs
  - Clients can compute data location
- Rule-based configuration
  - Desired/required replica count
  - Affinity/distribution rules
  - Infrastructure topology
  - Weighting
- Excellent data distribution
  - De-clustered placement
  - Excellent data-re-distribution
  - Migration proportional to change

CRUSH is by essence the crown jewel of the Ceph Storage Cluster.

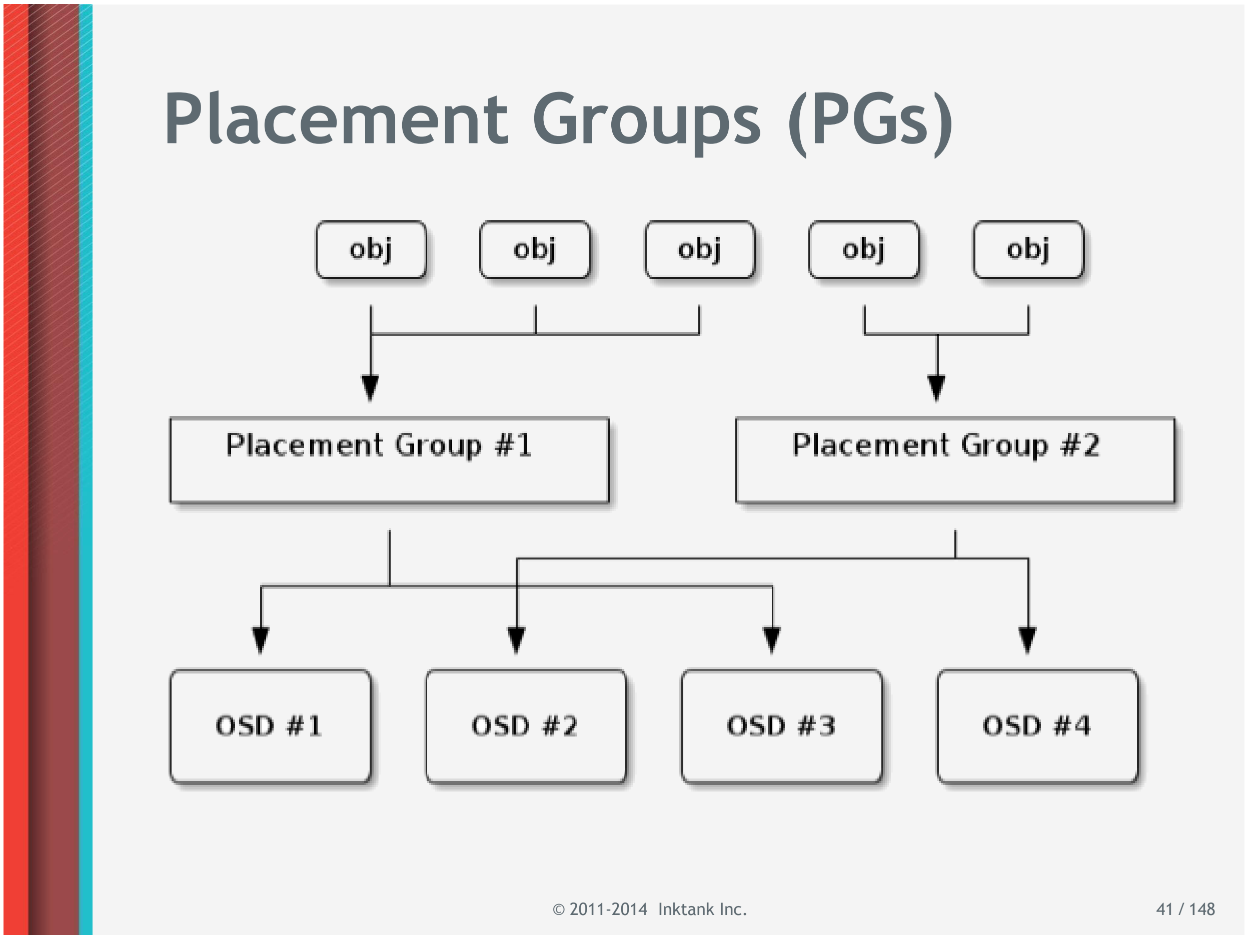
The OSD will make its decisions based on the CRUSH map it holds and will adapt when it receives a new one.



Placement Groups (PGs)

- What is a PG?
  - A PG is a logical collection of objects
  - Objects within the PG are replicated by the same set of devices
- How to choose the PG?
  - With CRUSH, data is first split into a certain number od sections
  - Each section□s called □placement groups□ (PG).
  - An object□s PG is determined by CRUSH
  - Choice is cased on
    - the hash of the object name,
    - the desired level of replication
    - the total number of placement groups in the system

A Placement Group (PG) aggregates a series of objects into a group, and maps the group to a series of OSDs.



Placement Groups (PGs)

obj

obj

obj

obj

obj

Placement Group #1

Placement Group #2

OSD #1

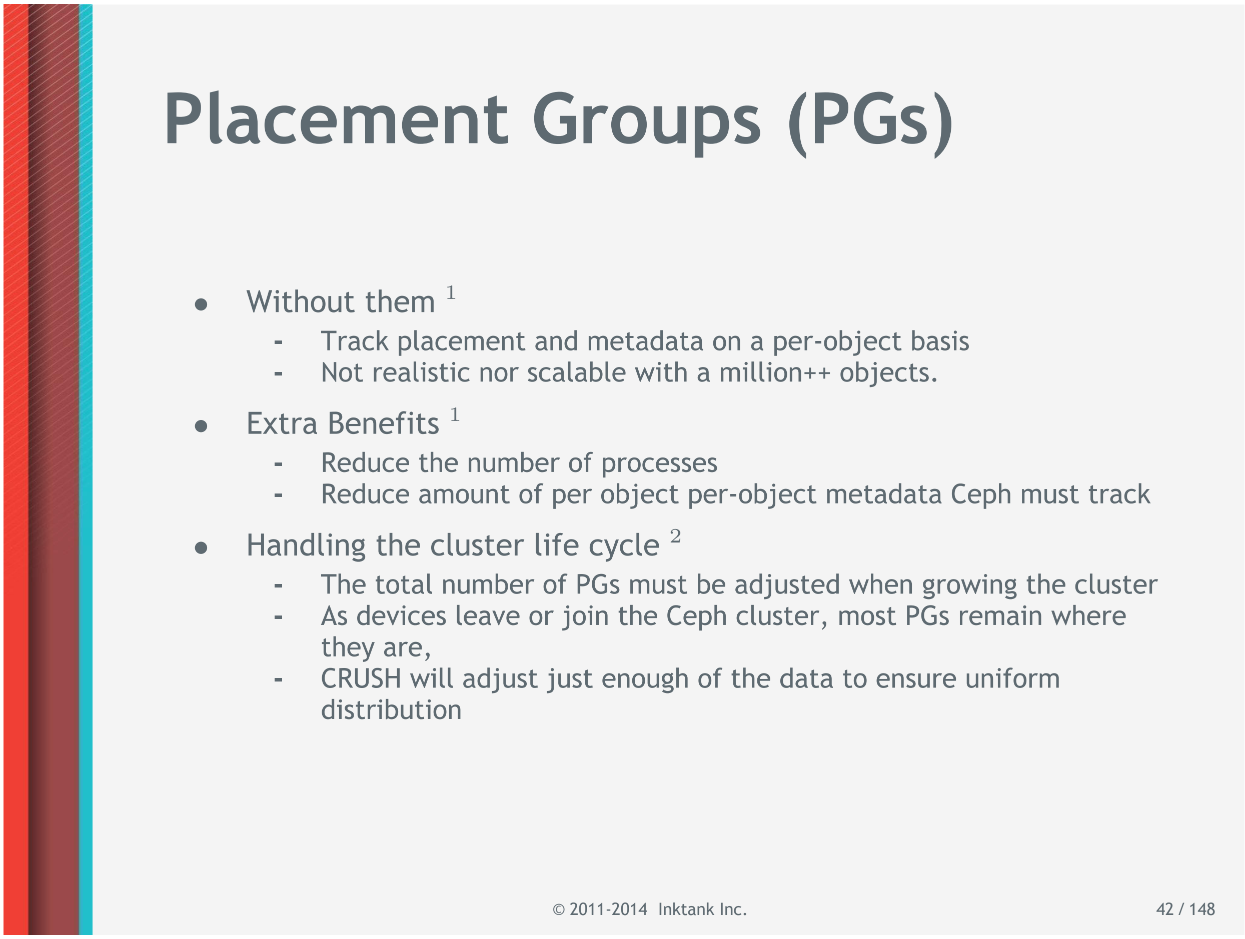
OSD #2

OSD #3

OSD #4

41 / 148

A Placement Group (PG) aggregates a series of objects onto a group, and maps the group to a series of OSDs.



Placement Groups (PGs)

- Without them [1]
  - Track placement and metadata on a per-object basis
  - Not realistic nor scalable with a million++ objects.
- Extra Benefits [1]
  - Reduce the number of processes
  - Reduce amount of per object per-object metadata Ceph must track
- Handling the cluster life cycle [2]
  - The total number of PGs must be adjusted when growing the cluster
  - As devices leave or join the Ceph cluster, most PGs remain where they are,
  - CRUSH will adjust just enough of the data to ensure uniform distribution

42 / 148

Note 1: Tracking object placement and object metadata on a per-object basis is computationally expensive-i.e., a system with millions of objects cannot realistically track placement on a per-object basis. Placement groups address this barrier to performance and scalability. Additionally, placement groups reduce the number of processes and the amount of per-object metadata Ceph must track when storing and retrieving data.

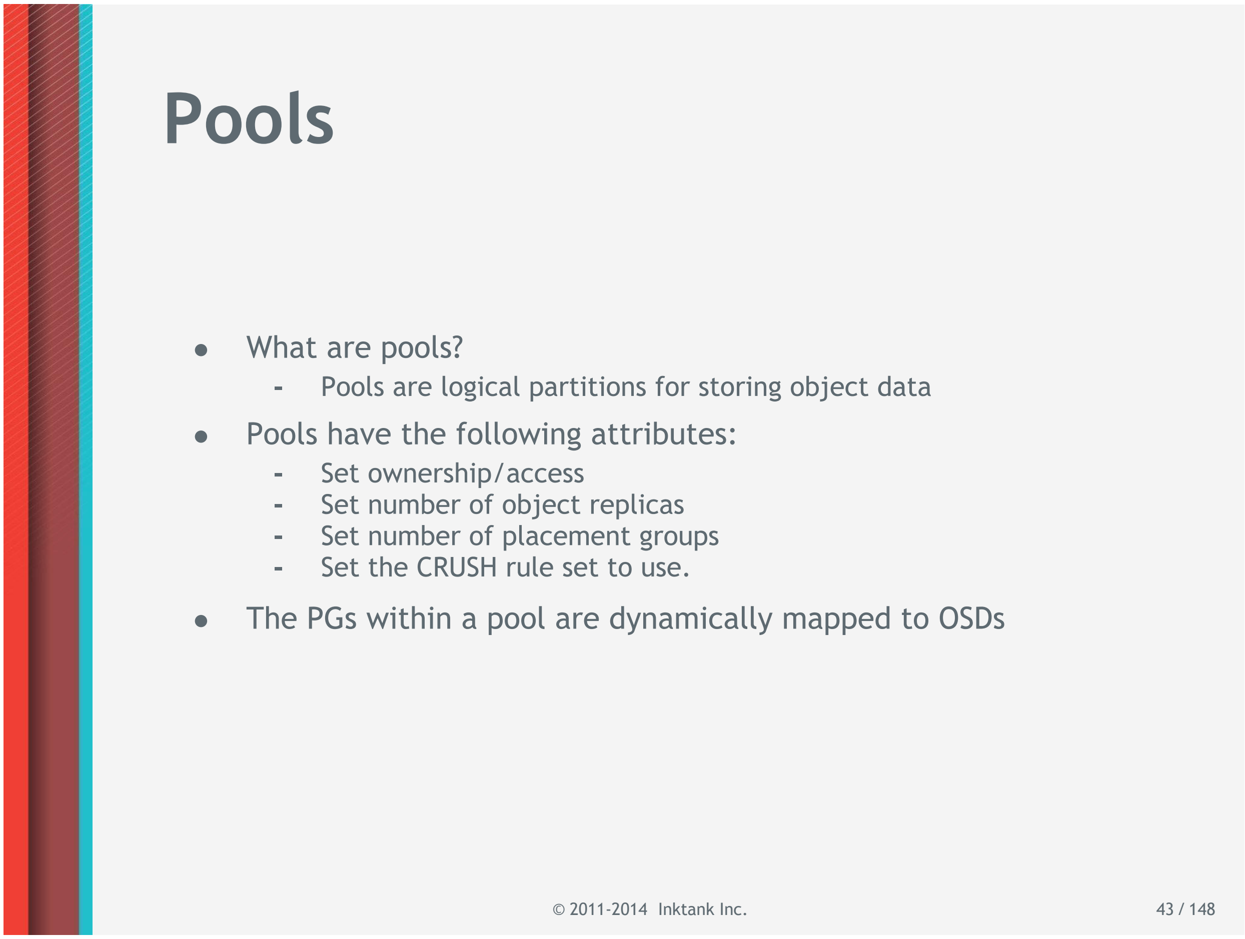
Note 2: Increasing the number of placement groups reduces the variance in per-OSD load across you cluster. We recommend approximately 50-200 placement groups per OSD to balance out memory and CPU requirements and per-OSD load. For a single pool of objects, you can use the following formula: Total Placement Groups = (OSDs*(50-200))/Number of replica.

When using multiple data pools for storing objects, you need to ensure that you balance the number of placement groups per pool with the number of placement groups per OSD so that you arrive at a reasonable total number of placement groups that provides reasonably low variance per OSD without taxing system resources or making the peering process too slow

1. `ceph osd pool set <pool-name> pg_num <pg_num>`
2. `ceph osd pool set <pool-name> pgp_num <pgp_num>`

The pgp_num parameter should be equal to pg_num

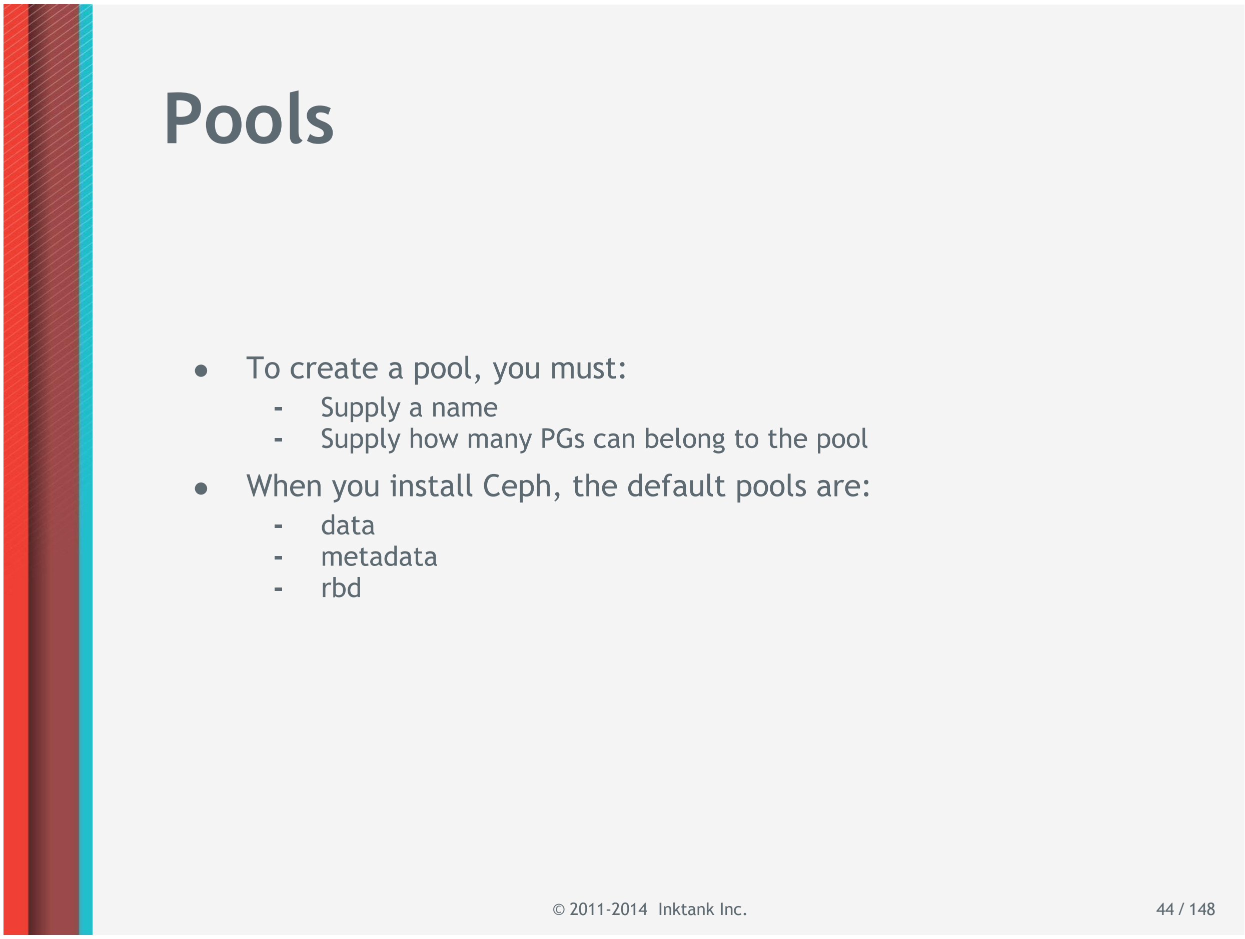
The second command will trigger the rebalancing of your data



Pools

- What are pools?
  - Pools are logical partitions for storing object data
- Pools have the following attributes:
  - Set ownership/access
  - Set number of object replicas
  - Set number of placement groups
  - Set the CRUSH rule set to use.
- The PGs within a pool are dynamically mapped to OSDs

When you first deploy a cluster without creating a pool, Ceph uses the default pools for storing data.
A pool has a default number of replica. Currently 2 but the Firefly version will bump the default up to 3.
A pool differs from CRUSH¤s location-based buckets in that a pool doesn¤t have a single physical location, and a pool provides you with some additional functionality, including: Replicas: You can set the desired number of copies/replicas of an object
- A typical configuration stored an object and one additional copy (i.e., size = 2), but you can determine the number of copies/replicas.
Placement Groups: you can set the number of placement groups for the pool.
- A typical configuration uses approximately 100 placement groups per OSD to provide optimal balancing without using up too many computing resources. When setting up multiple pools, be careful to ensure you set a reasonable number of placement groups for both the pool and the cluster as a whole.
CRUSH Rules: When you store data in a pool, a CRUSH rule set mapped to the pool enables CRUSH
- To identify a rule for the placement of the primary object and object replicas in your cluster. You can create a custom CRUSH rule for your pool.
Snapshots: When you create snapshots with `ceph osd pool mksnap`, you effectively take a snapshot of a particular pool.
Set Ownership: You can set a user ID as the owner of a pool.



Pools

- To create a pool, you must:
  - Supply a name
  - Supply how many PGs can belong to the pool
- When you install Ceph, the default pools are:
  - data
  - metadata
  - rbd

© 2011-2014 Inktank Inc.

44 / 148

To organize data into pools, you can list, create, and remove pools. You can also view the utilization statistics for each pool.

Listing the pools:

```
  ceph osd lspools
```

Creating the pools:

```
  ceph osd pool create {pool-name} {pg-num} [{pgp-num}]
```

Deleting the pools:

```
  ceph osd pool delete {pool-name} [{pool-name} --yes-i-really-really-mean-it]
```

Renaming the pools:

```
  ceph osd pool rename {current-pool-name} {new-pool-name}
```

Statistics for the pools:

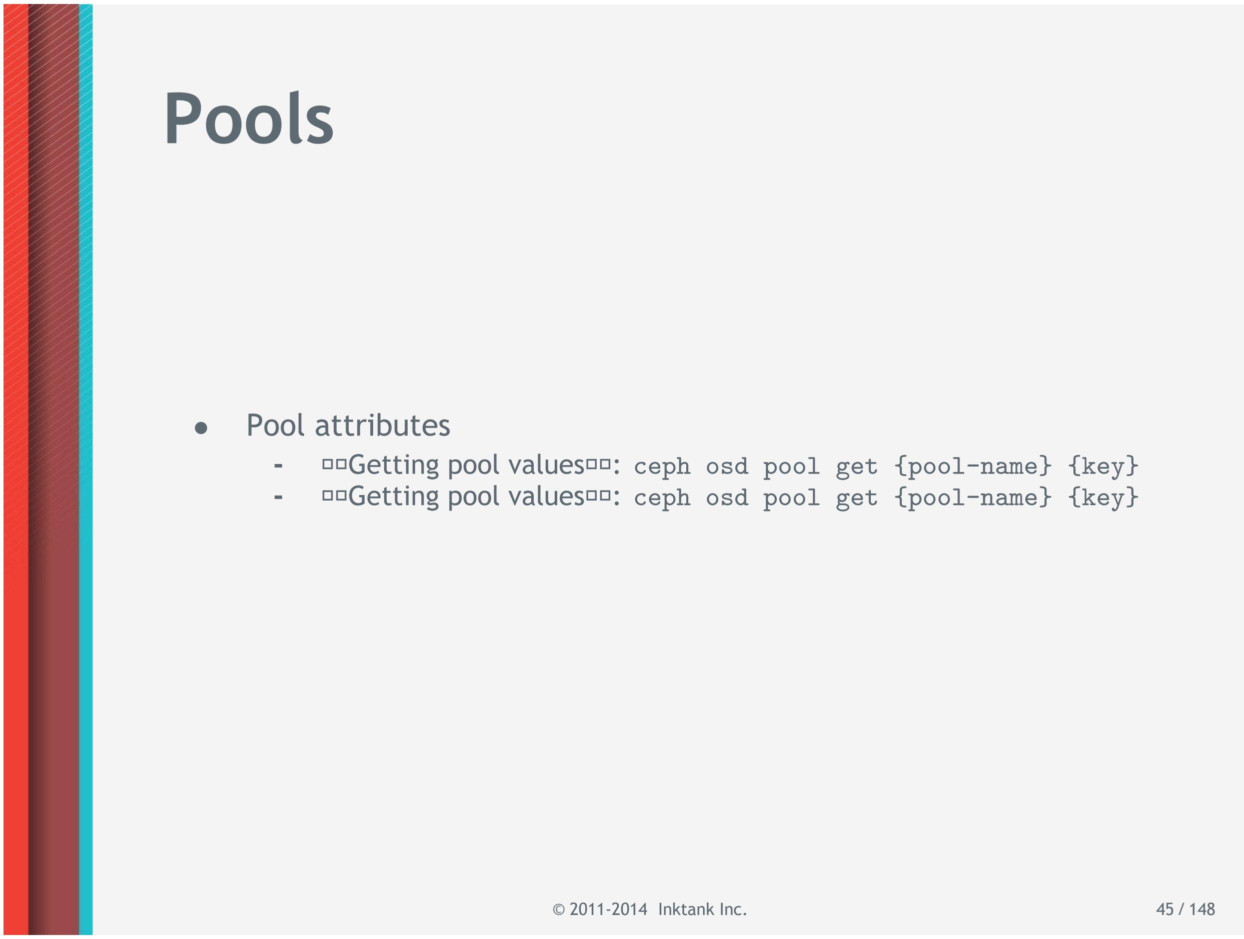
```
  rados df
```

Snapshotting pools:

```
  ceph osd pool mksnap {pool-name} {snap-name}
```

Removing a snapshot:

```
  ceph osd pool rmsnap {pool-name} {snap-name}
```



Pools

- Pool attributes
  - □□Getting pool values□□: `ceph osd pool get {pool-name} {key}`
  - □□Getting pool values□□: `ceph osd pool get {pool-name} {key}`

Attributes

`size`: number of replica objects

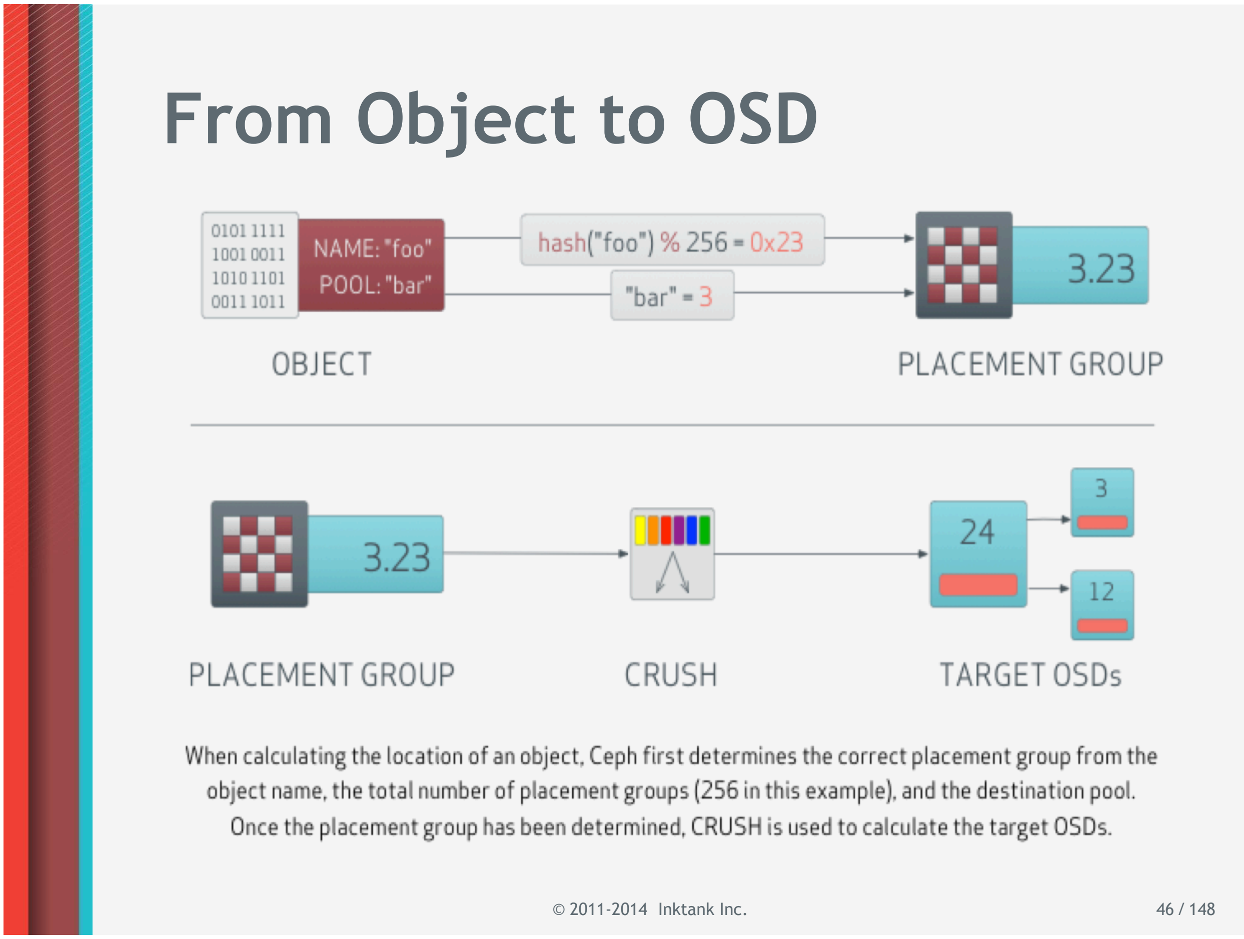
`min_size`: minimum number of replica available for IO

`crash_replay_interval`: number of seconds to allow clients to replay acknowledged, but uncommitted requests

`pgp_num`: effective number of placement groups to use when calculating data placement

`crush_ruleset`: ruleset to use for mapping object placement in the cluster (CRUSH Map Module)

`hashpspool`: get HASHPSPOOL flag on a given pool



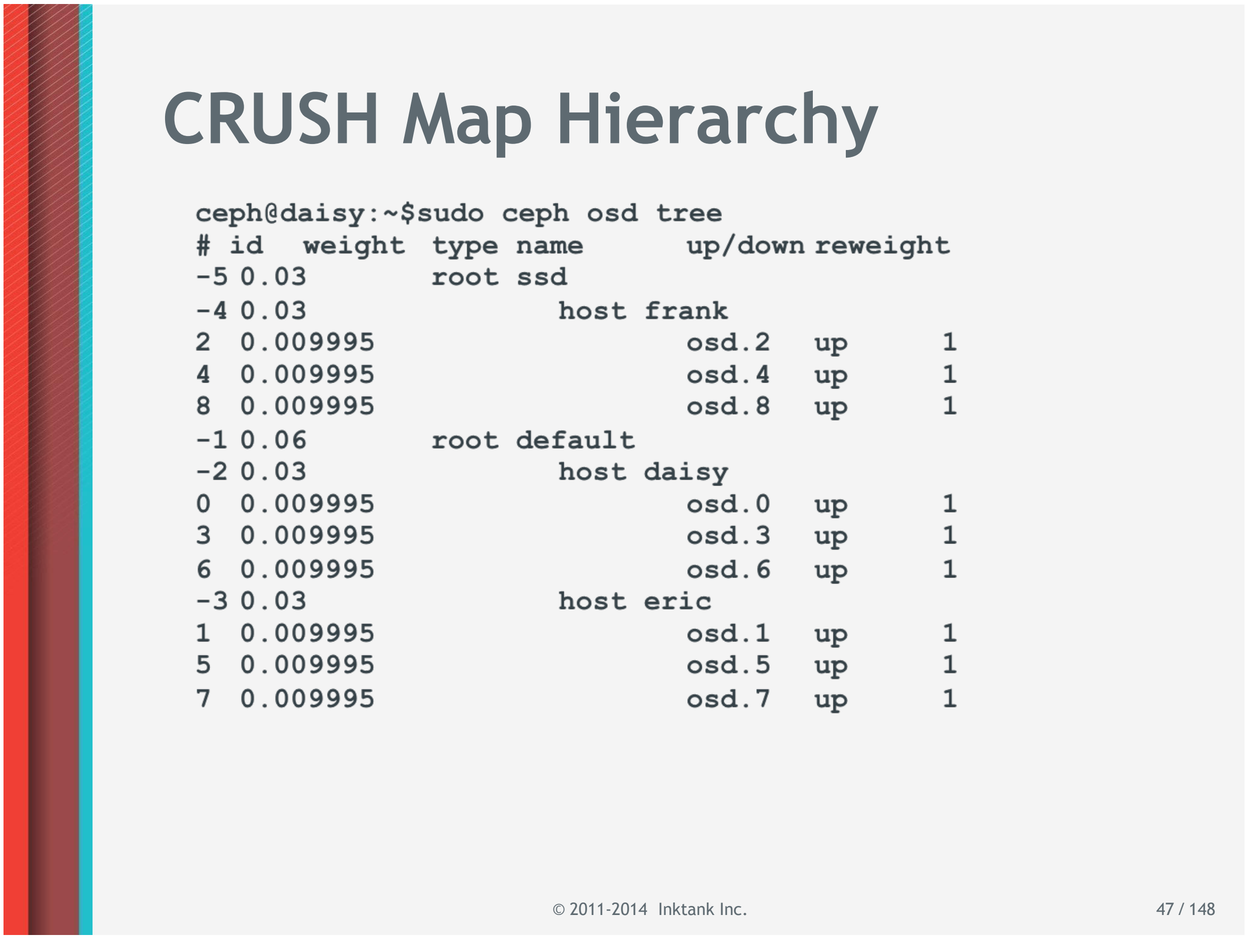
From Object to OSD

When calculating the location of an object, Ceph first determines the correct placement group from the object name, the total number of placement groups (256 in this example), and the destination pool. Once the placement group has been determined, CRUSH is used to calculate the target OSDs.

46 / 148

To generate the PG id, we use - The pool id - A hashing formula based on the object name modulo the number of PGs

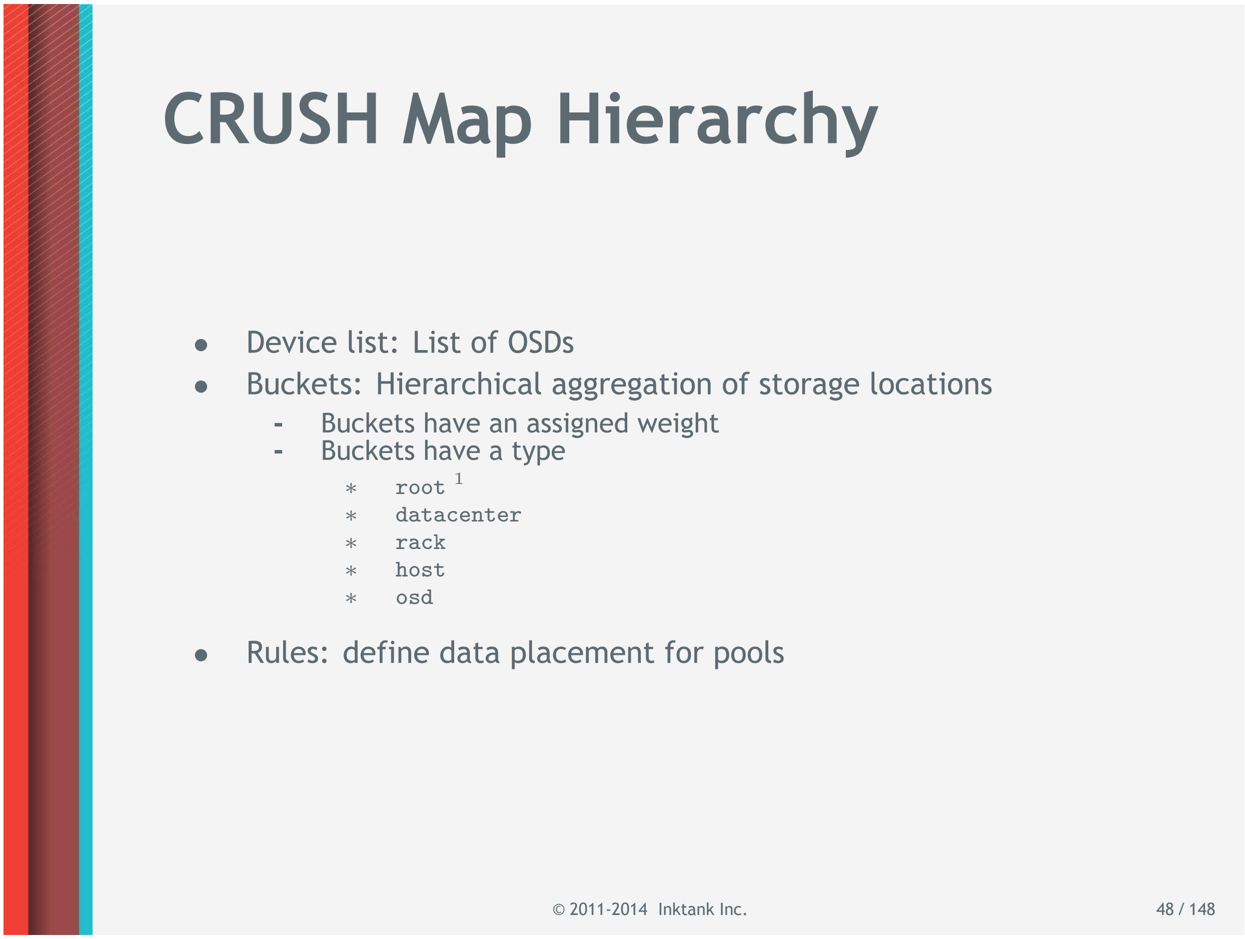
First OSD in the list returned is the primary OSD, the next ones are secondary



CRUSH Map Hierarchy

```
ceph@daisy:~$sudo ceph osd tree
# id  weight  type name      up/down reweight
-5 0.03       root ssd
-4 0.03              host frank
2  0.009995                  osd.2  up      1
4  0.009995                  osd.4  up      1
8  0.009995                  osd.8  up      1
-1 0.06       root default
-2 0.03              host daisy
0  0.009995                  osd.0  up      1
3  0.009995                  osd.3  up      1
6  0.009995                  osd.6  up      1
-3 0.03              host eric
1  0.009995                  osd.1  up      1
5  0.009995                  osd.5  up      1
7  0.009995                  osd.7  up      1
```

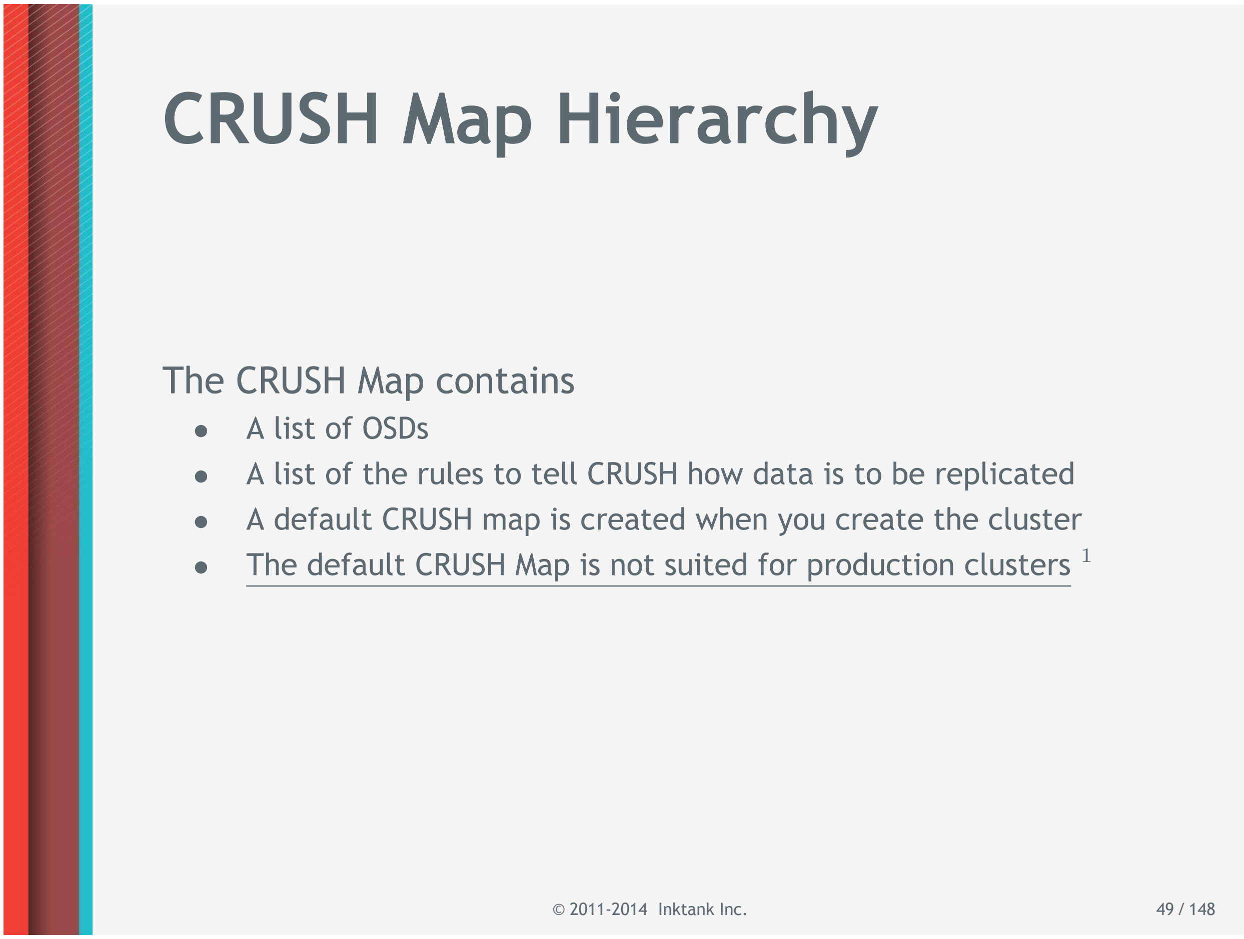
The command used to view the CRUSH Map is: `ceph osd tree`



CRUSH Map Hierarchy

- Device list: List of OSDs
- Buckets: Hierarchical aggregation of storage locations
  - Buckets have an assigned weight
  - Buckets have a type
    - `root` [1]
    - `datacenter`
    - `rack`
    - `host`
    - `osd`
- Rules: define data placement for pools

Note 1: Ceph pool

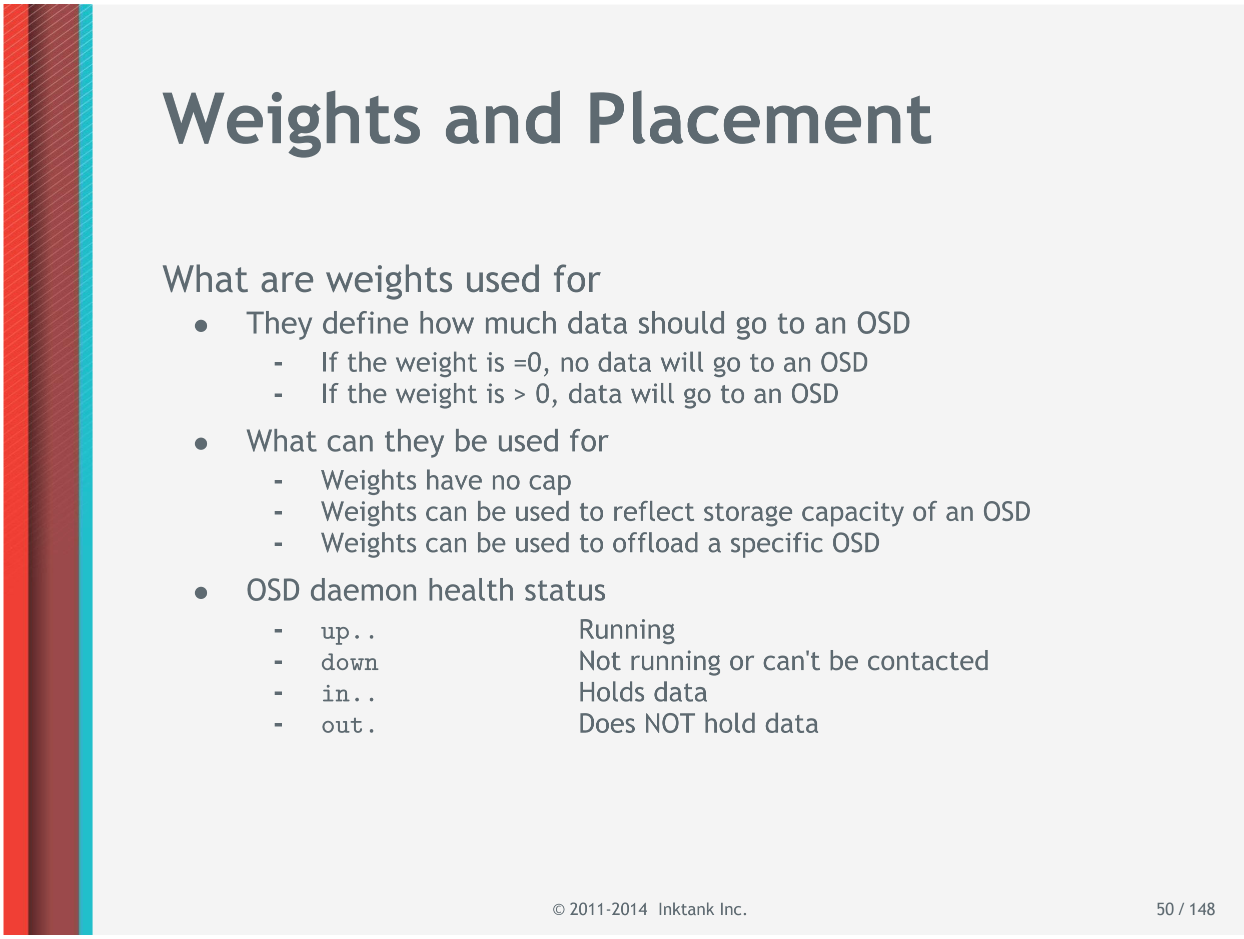


CRUSH Map Hierarchy

The CRUSH Map contains

- A list of OSDs
- A list of the rules to tell CRUSH how data is to be replicated
- A default CRUSH map is created when you create the cluster
- The default CRUSH Map is not suited for production clusters [1]

Note 1: This default CRUSH Map is fine for a sandbox-type installation only! For production clusters, it should be customized for better management, performance, and data security



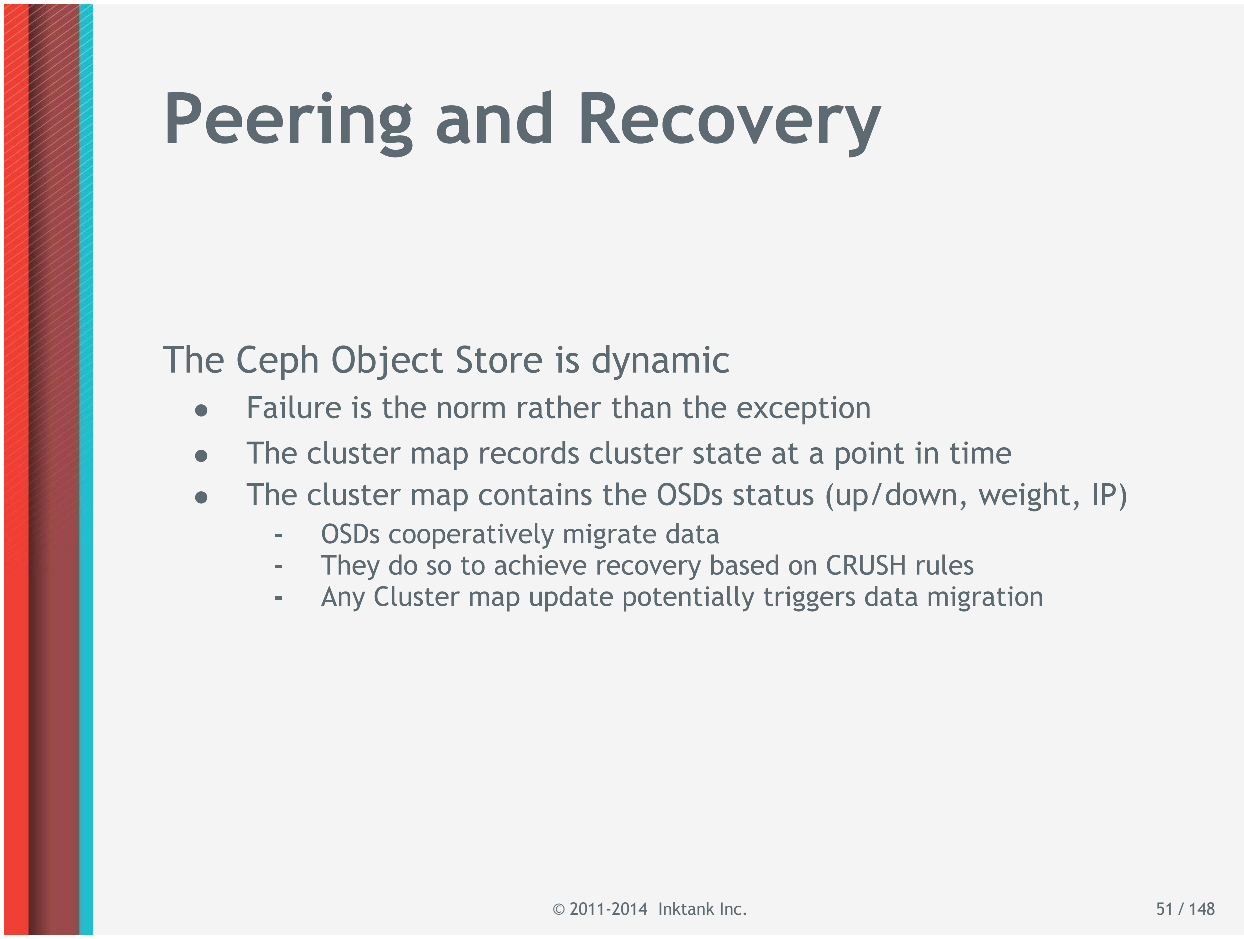
Weights and Placement

What are weights used for

- They define how much data should go to an OSD
  - If the weight is =0, no data will go to an OSD
  - If the weight is > 0, data will go to an OSD
- What can they be used for
  - Weights have no cap
  - Weights can be used to reflect storage capacity of an OSD
  - Weights can be used to offload a specific OSD
- OSD daemon health status

`up..`	Running
`down`	Not running or can't be contacted
`in..`	Holds data
`out.`	Does NOT hold data

As a quick way to remember it, the weight value indicates the proportion of data an OSD will hold if it is up and running



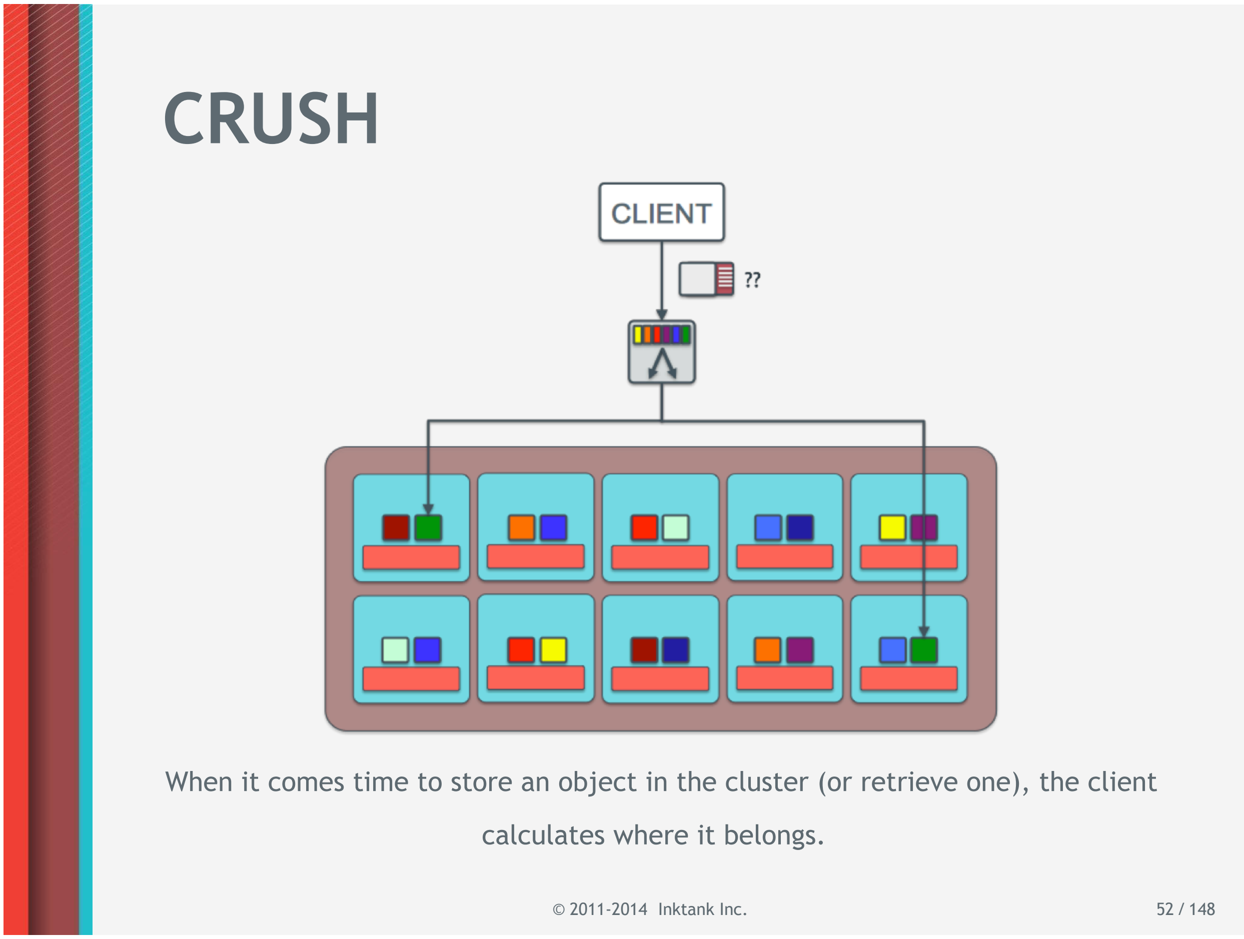
Peering and Recovery

The Ceph Object Store is dynamic

- Failure is the norm rather than the exception
- The cluster map records cluster state at a point in time
- The cluster map contains the OSDs status (up/down, weight, IP)
  - OSDs cooperatively migrate data
  - They do so to achieve recovery based on CRUSH rules
  - Any Cluster map update potentially triggers data migration

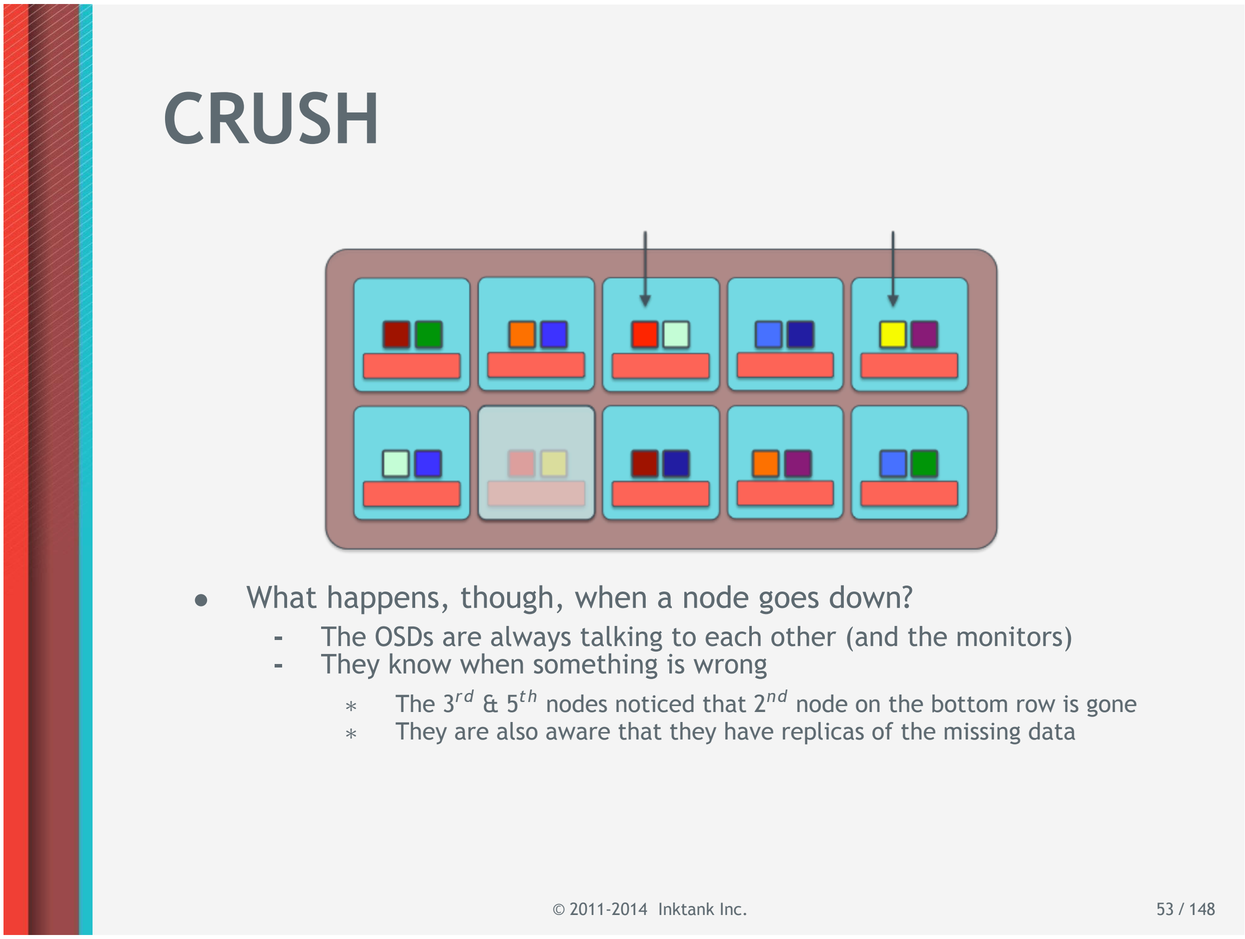
By default, if an OSD has been down for 5 minutes or more, we will start copying data to other OSDs in order to satisfy the number of replicas the pool must hold.

Remember that is the number of replica available goes below the min_size pool parameter, no IO will be served.



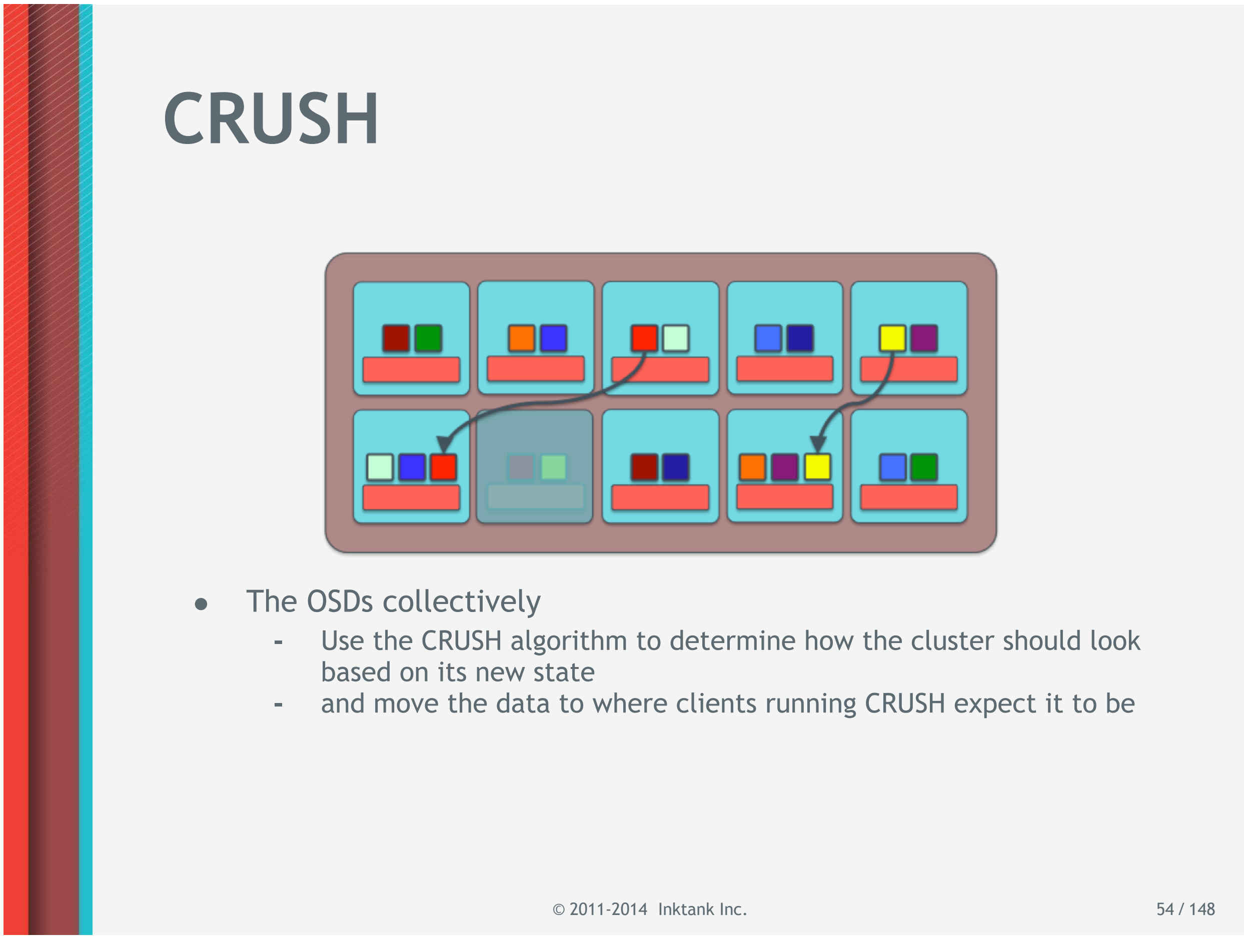
CRUSH

When it comes time to store an object in the cluster (or retrieve one), the client calculates where it belongs.



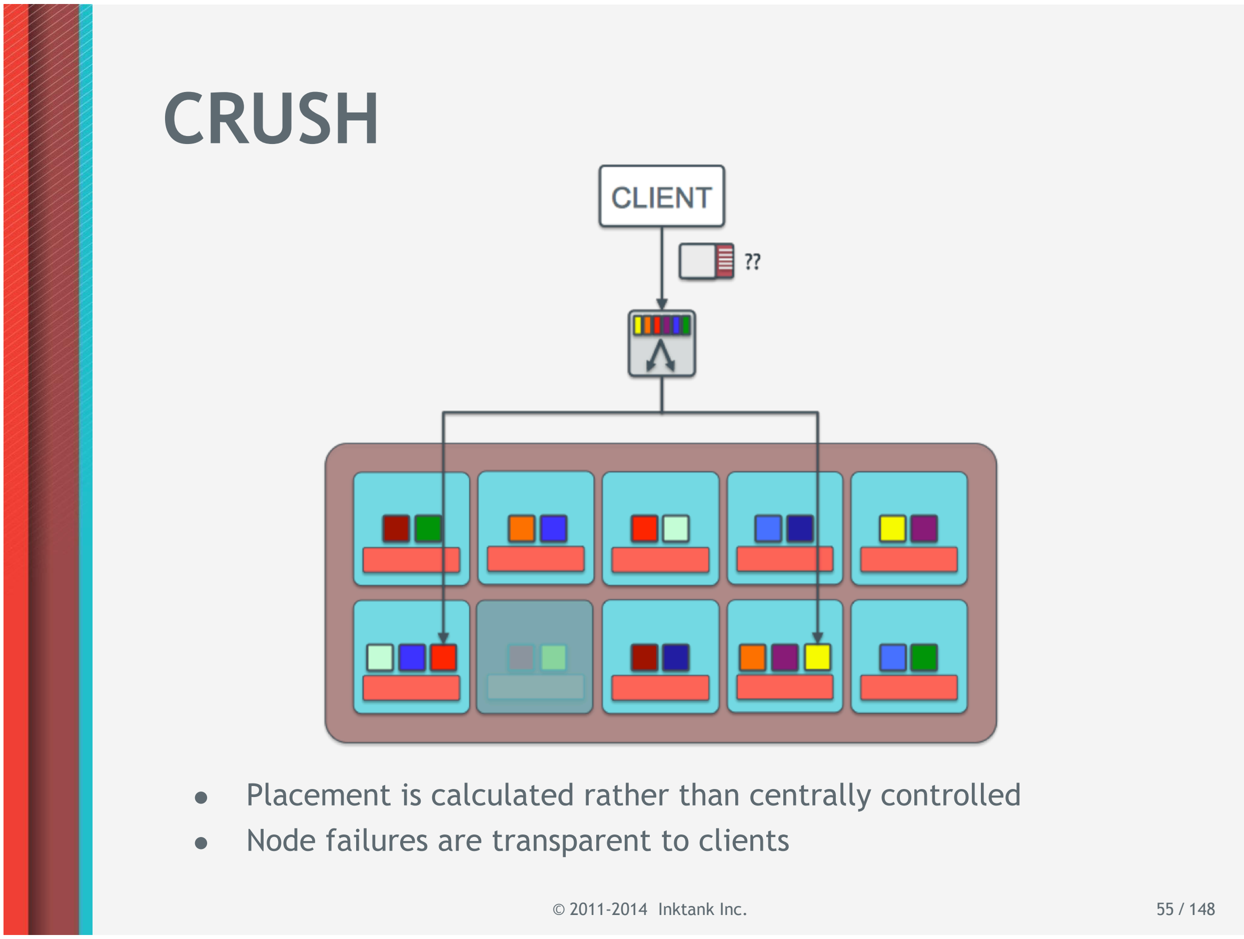
CRUSH

- What happens, though, when a node goes down?
  - The OSDs are always talking to each other (and the monitors)
  - They know when something is wrong
    - The 3rd & 5th nodes noticed that 2nd node on the bottom row is gone
    - They are also aware that they have replicas of the missing data



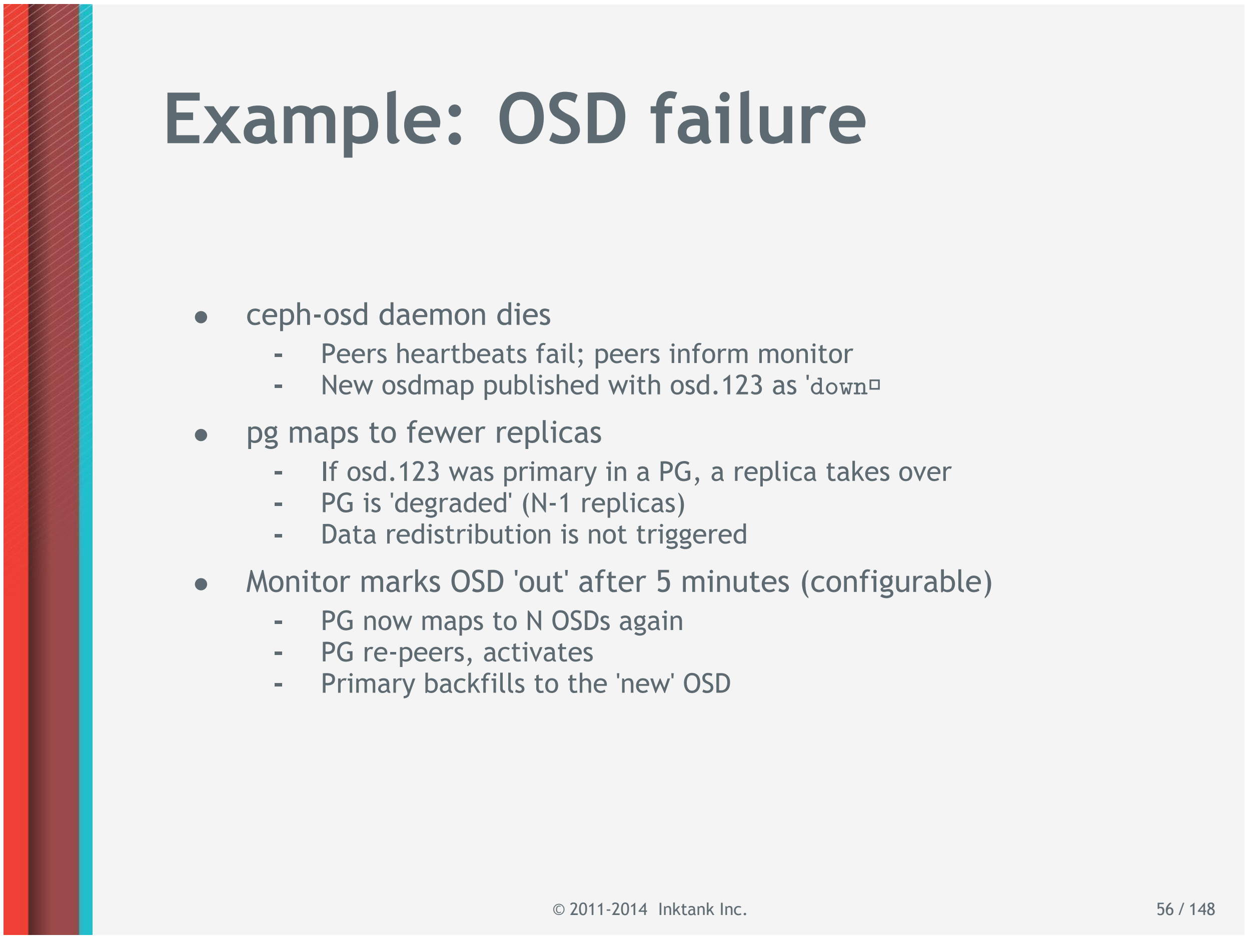
CRUSH

- The OSDs collectively
  - Use the CRUSH algorithm to determine how the cluster should look based on its new state
  - and move the data to where clients running CRUSH expect it to be



CRUSH

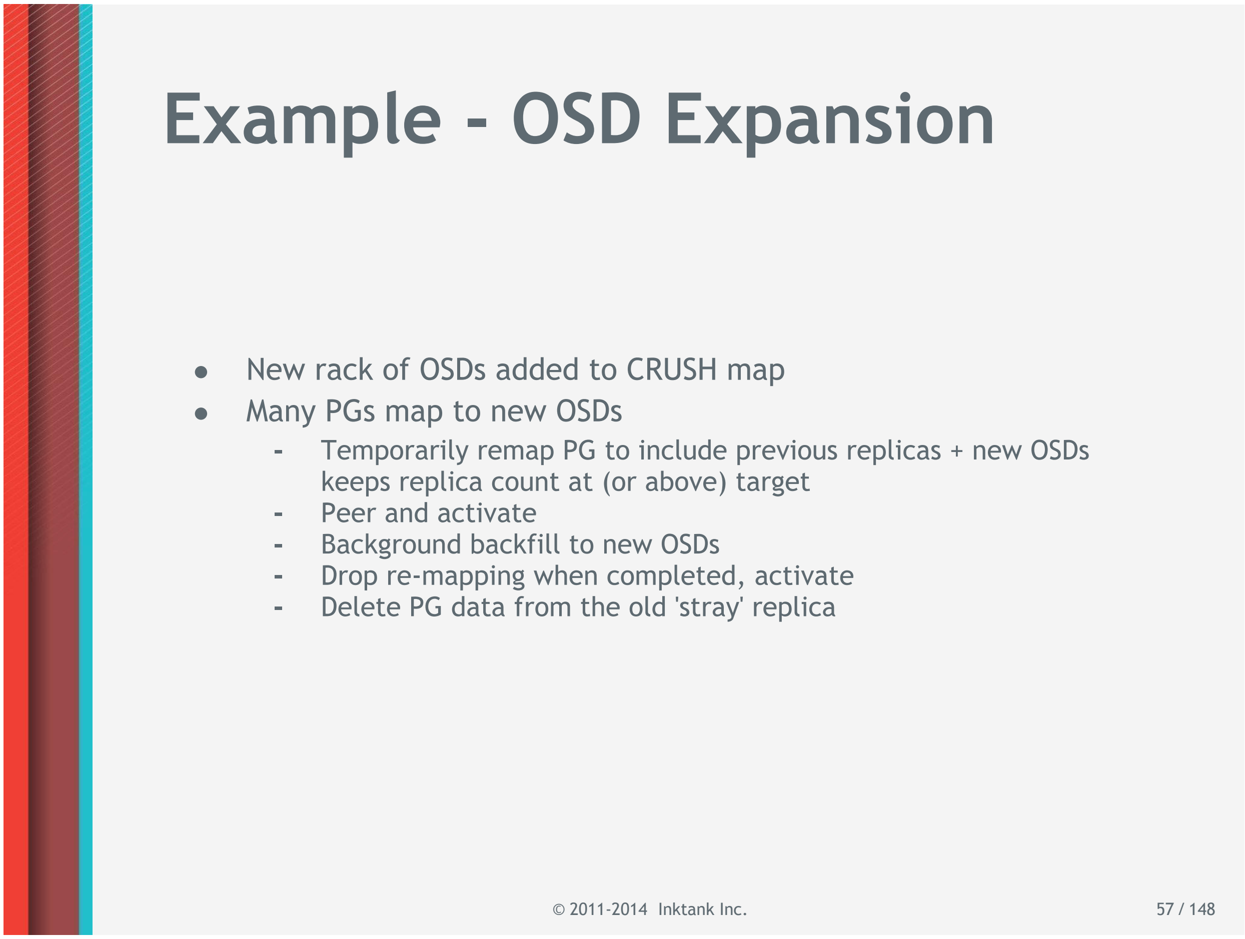
- Placement is calculated rather than centrally controlled
- Node failures are transparent to clients



Example: OSD failure

- ceph-osd daemon dies
  - Peers heartbeats fail; peers inform monitor
  - New osdmap published with osd.123 as 'down□
- pg maps to fewer replicas
  - If osd.123 was primary in a PG, a replica takes over
  - PG is 'degraded' (N-1 replicas)
  - Data redistribution is not triggered
- Monitor marks OSD 'out' after 5 minutes (configurable)
  - PG now maps to N OSDs again
  - PG re-peers, activates
  - Primary backfills to the 'new' OSD

Re-peering should be quick
Few seconds for 1 PG
Less than 30 seconds for 100 PGs
Less than 5 minutes for 1000 PGs
When this process takes place, remember to check the private network where copying/replicating takes place
You need a primary PG to satisfy a read or a write IO request.
The client will experience increased latency during the re-peering and before the new primary PG gets elected.
A PG being re-peered will not accept read and write operations

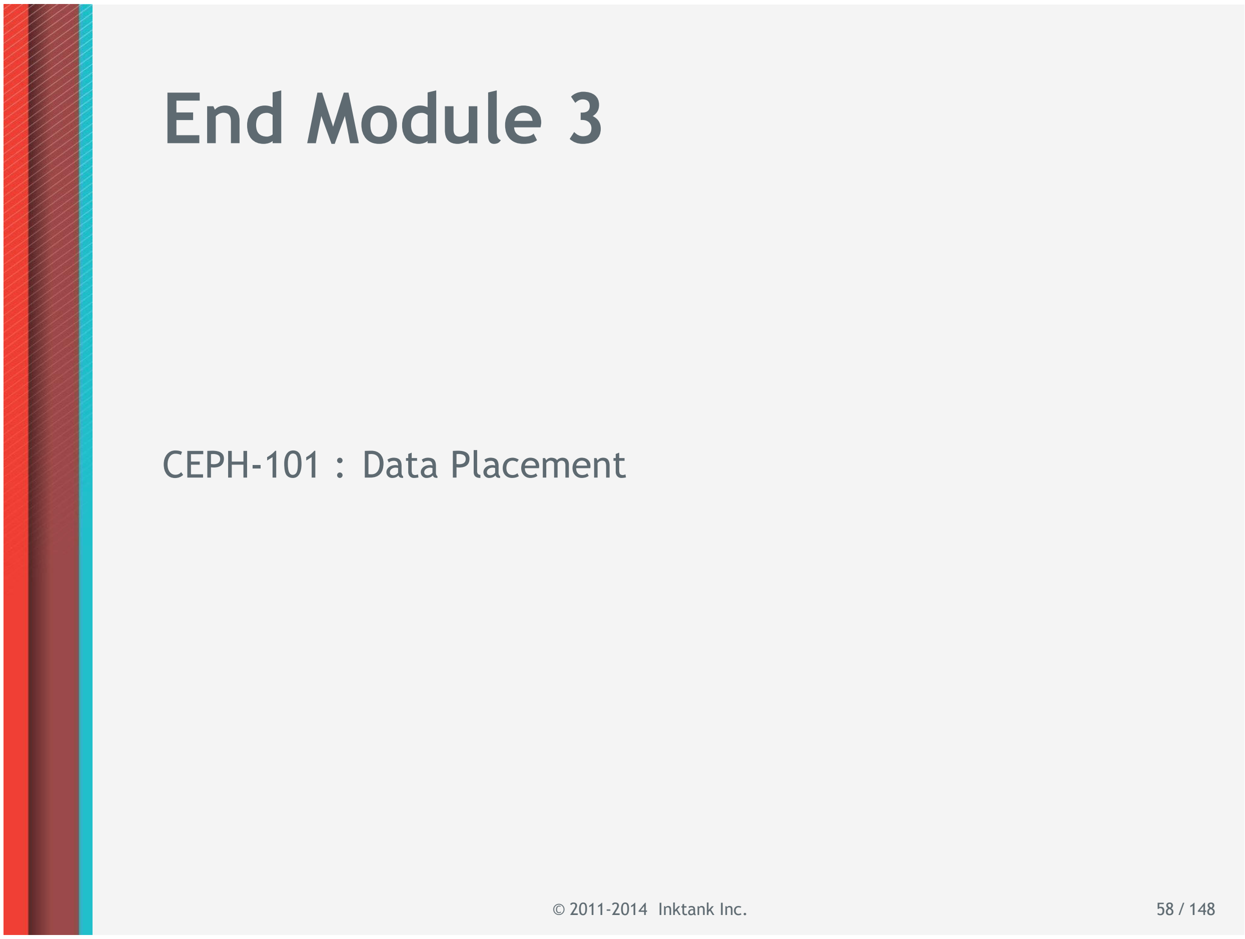


Example - OSD Expansion

- New rack of OSDs added to CRUSH map
- Many PGs map to new OSDs
  - Temporarily remap PG to include previous replicas + new OSDs keeps replica count at (or above) target
  - Peer and activate
  - Background backfill to new OSDs
  - Drop re-mapping when completed, activate
  - Delete PG data from the old 'stray' replica

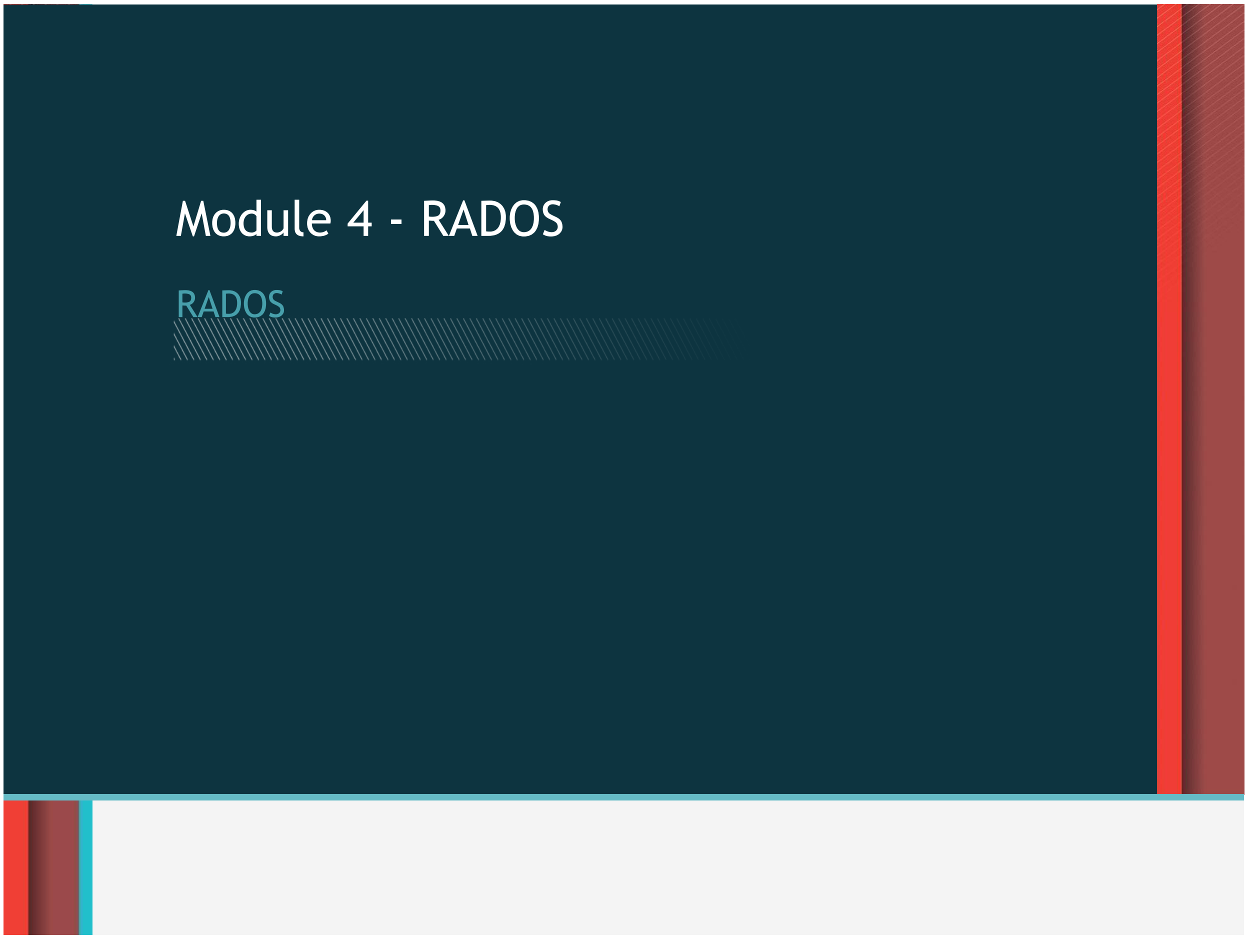
Remember the redistribution is proportional to the change introduced.

For a while, you will use extra space because of the copy that sits on the new OSDs plus the old copies that remains until it is disposed off when the backfill completes.



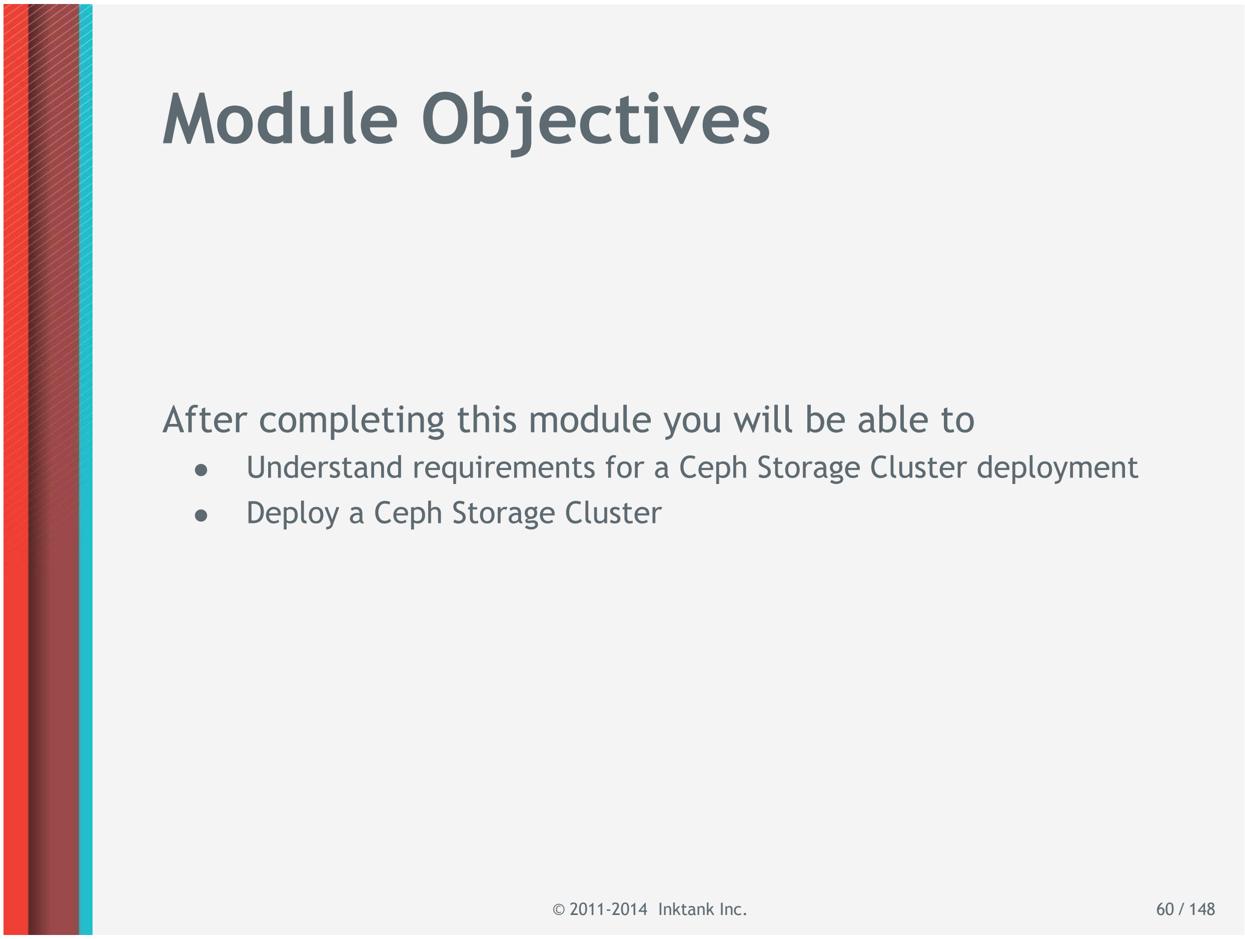
End Module 3

CEPH-101 : Data Placement



Module 4 - RADOS

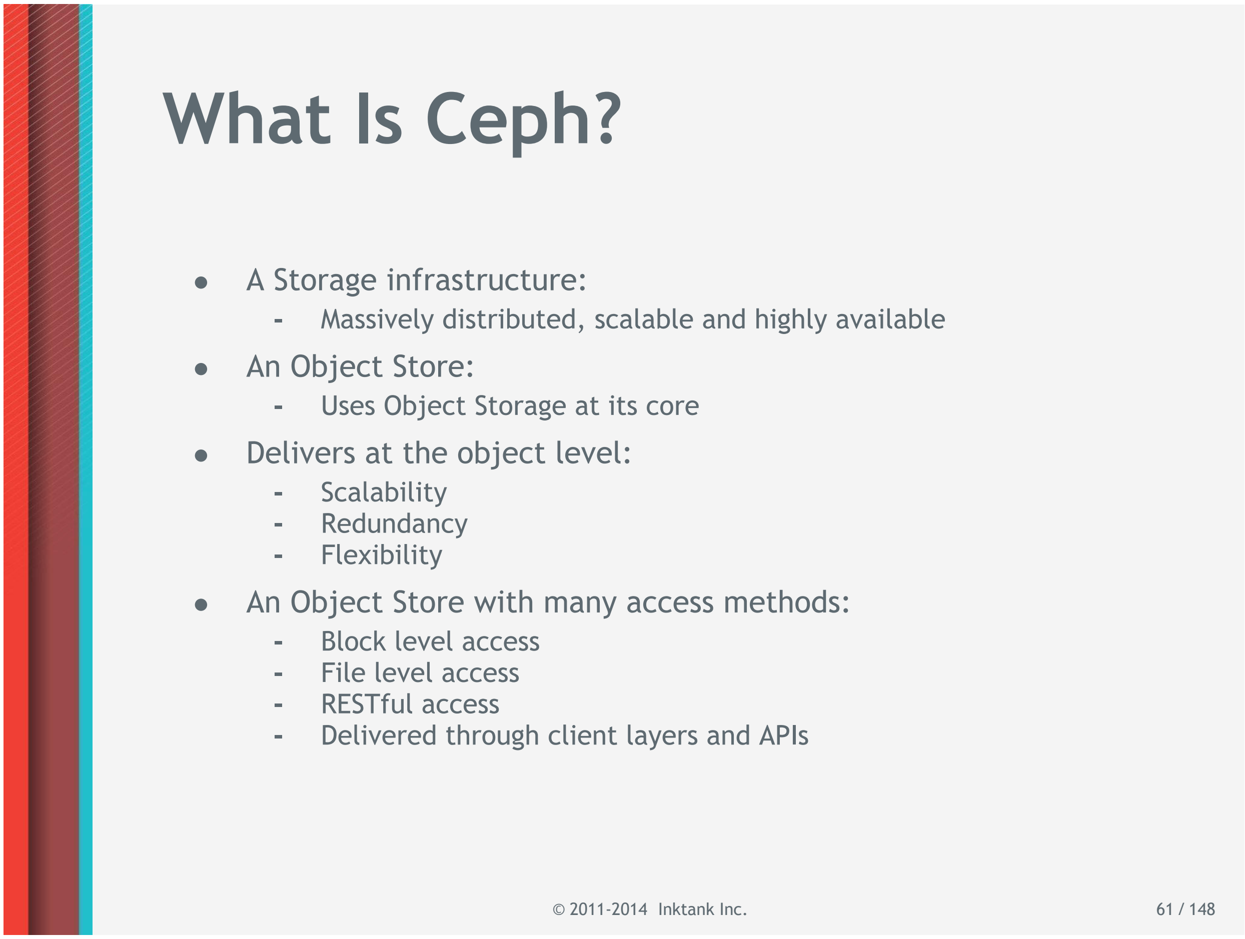
RADOS



Module Objectives

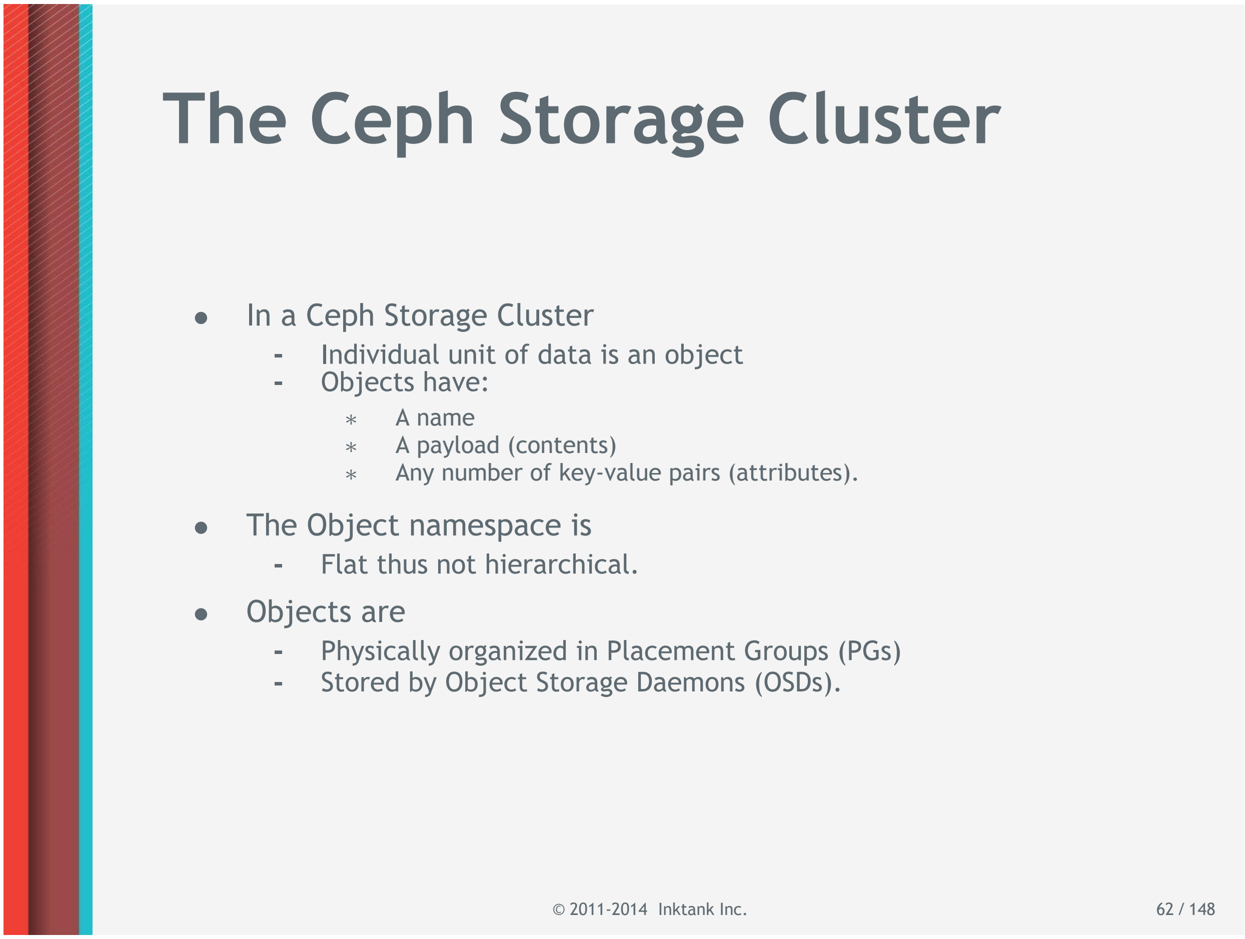
After completing this module you will be able to

- Understand requirements for a Ceph Storage Cluster deployment
- Deploy a Ceph Storage Cluster



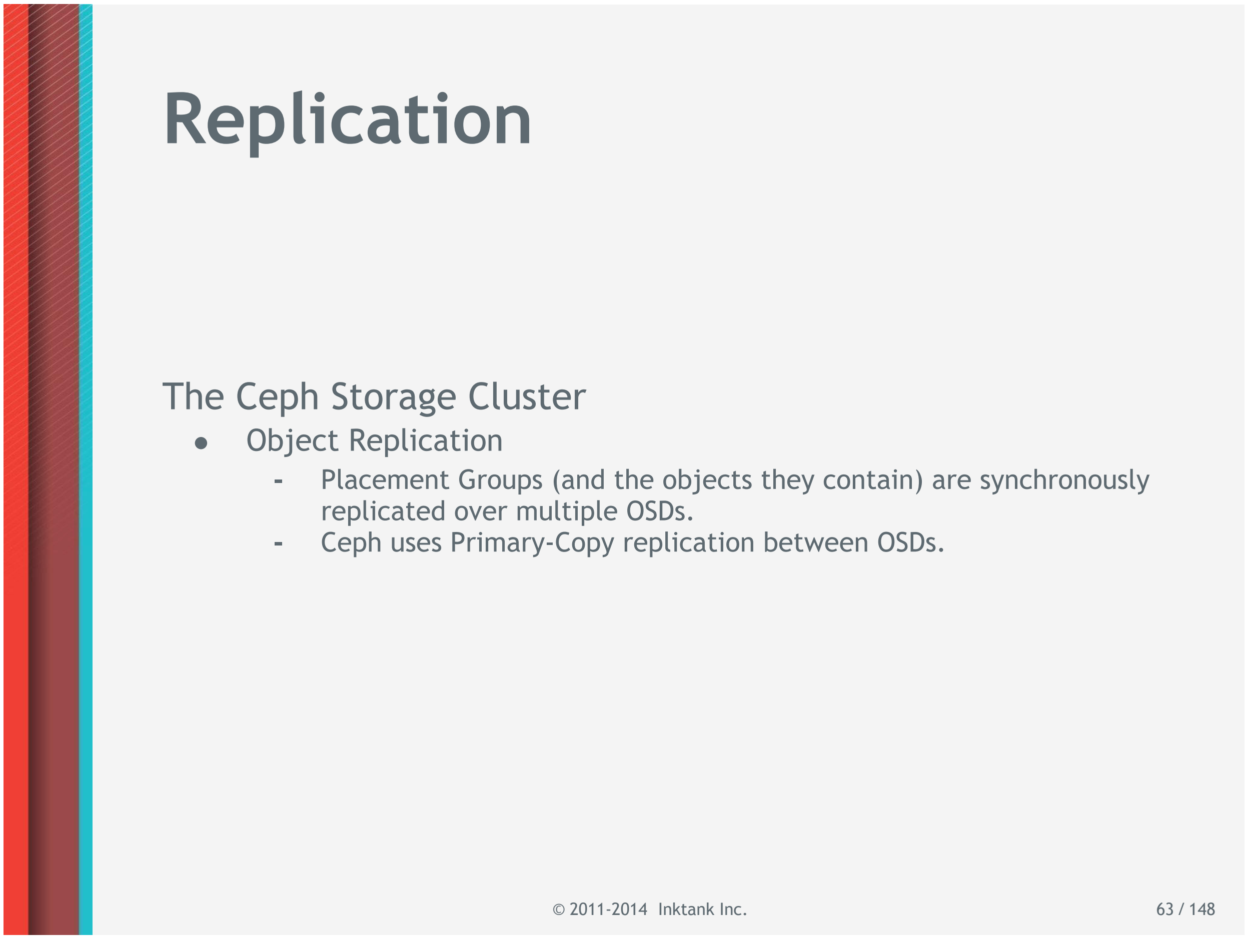
What Is Ceph?

- A Storage infrastructure:
  - Massively distributed, scalable and highly available
- An Object Store:
  - Uses Object Storage at its core
- Delivers at the object level:
  - Scalability
  - Redundancy
  - Flexibility
- An Object Store with many access methods:
  - Block level access
  - File level access
  - RESTful access
  - Delivered through client layers and APIs



The Ceph Storage Cluster

- In a Ceph Storage Cluster
  - Individual unit of data is an object
  - Objects have:
    - A name
    - A payload (contents)
    - Any number of key-value pairs (attributes).
- The Object namespace is
  - Flat thus not hierarchical.
- Objects are
  - Physically organized in Placement Groups (PGs)
  - Stored by Object Storage Daemons (OSDs).

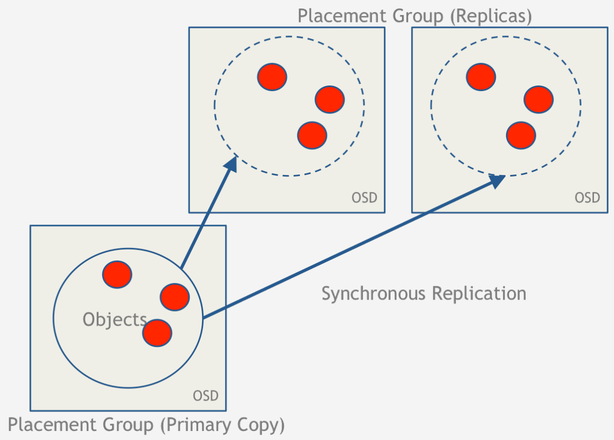


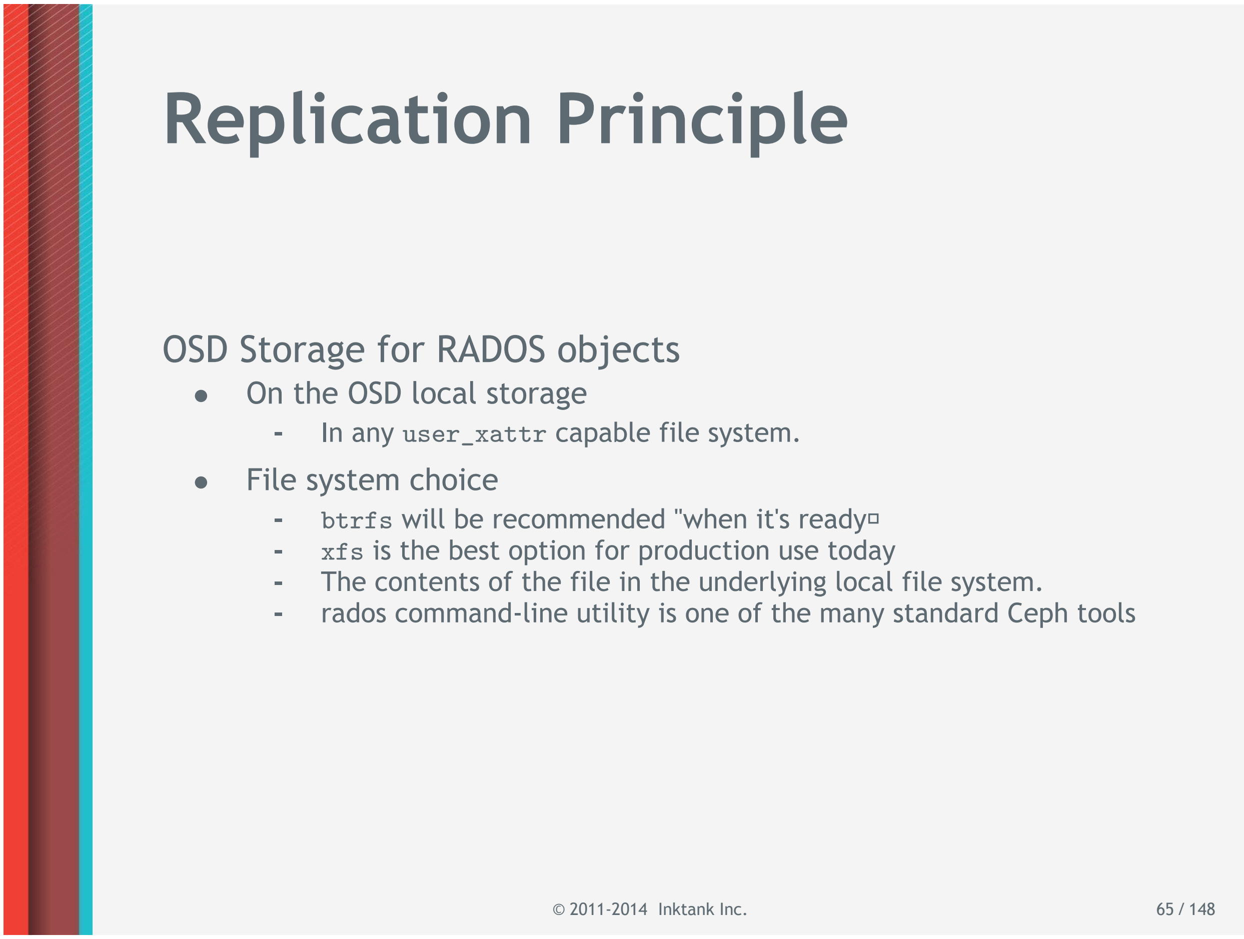
Replication

The Ceph Storage Cluster

- Object Replication
    - Placement Groups (and the objects they contain) are synchronously replicated over multiple OSDs.
    - Ceph uses Primary-Copy replication between OSDs.

Object Replication Principle

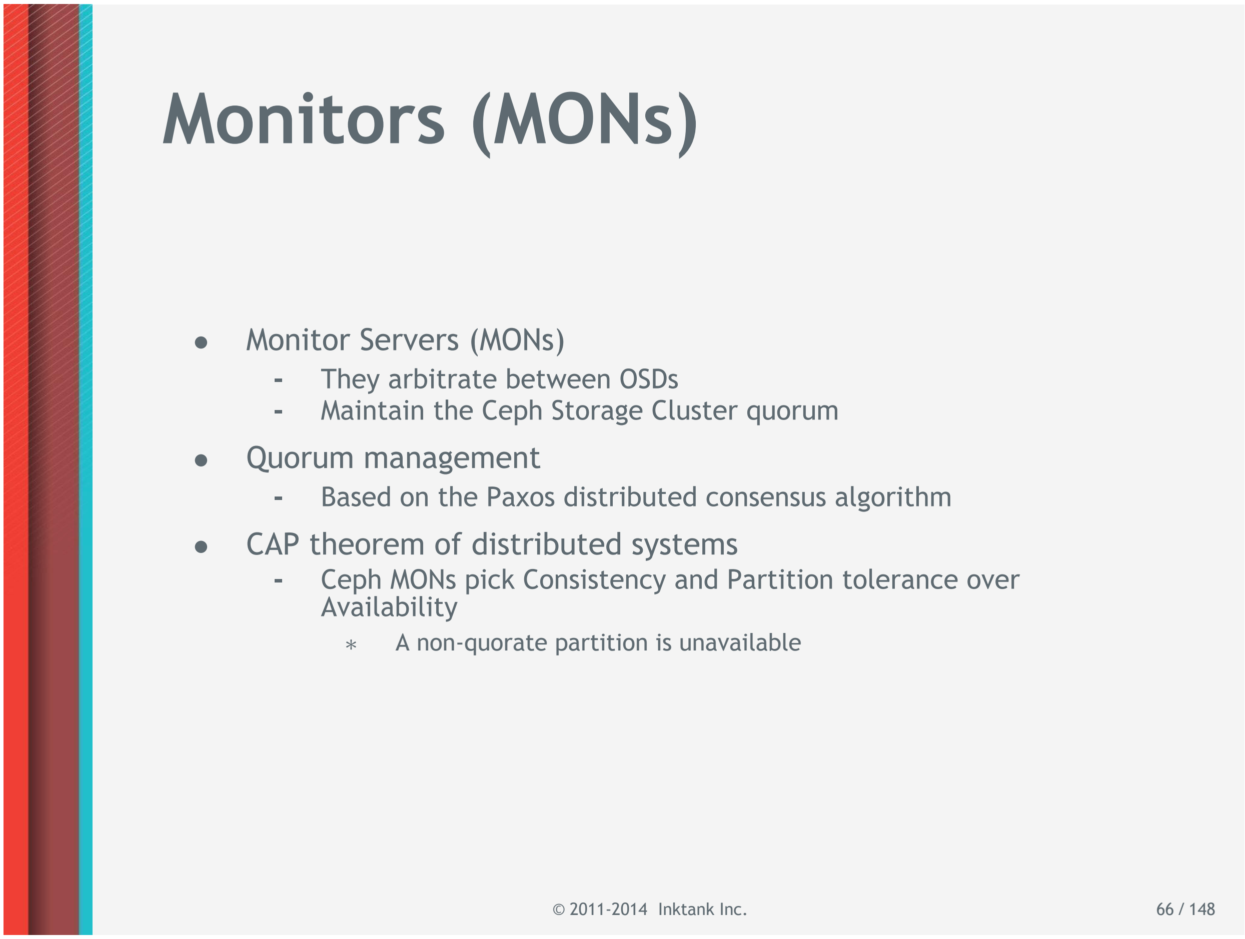




Replication Principle

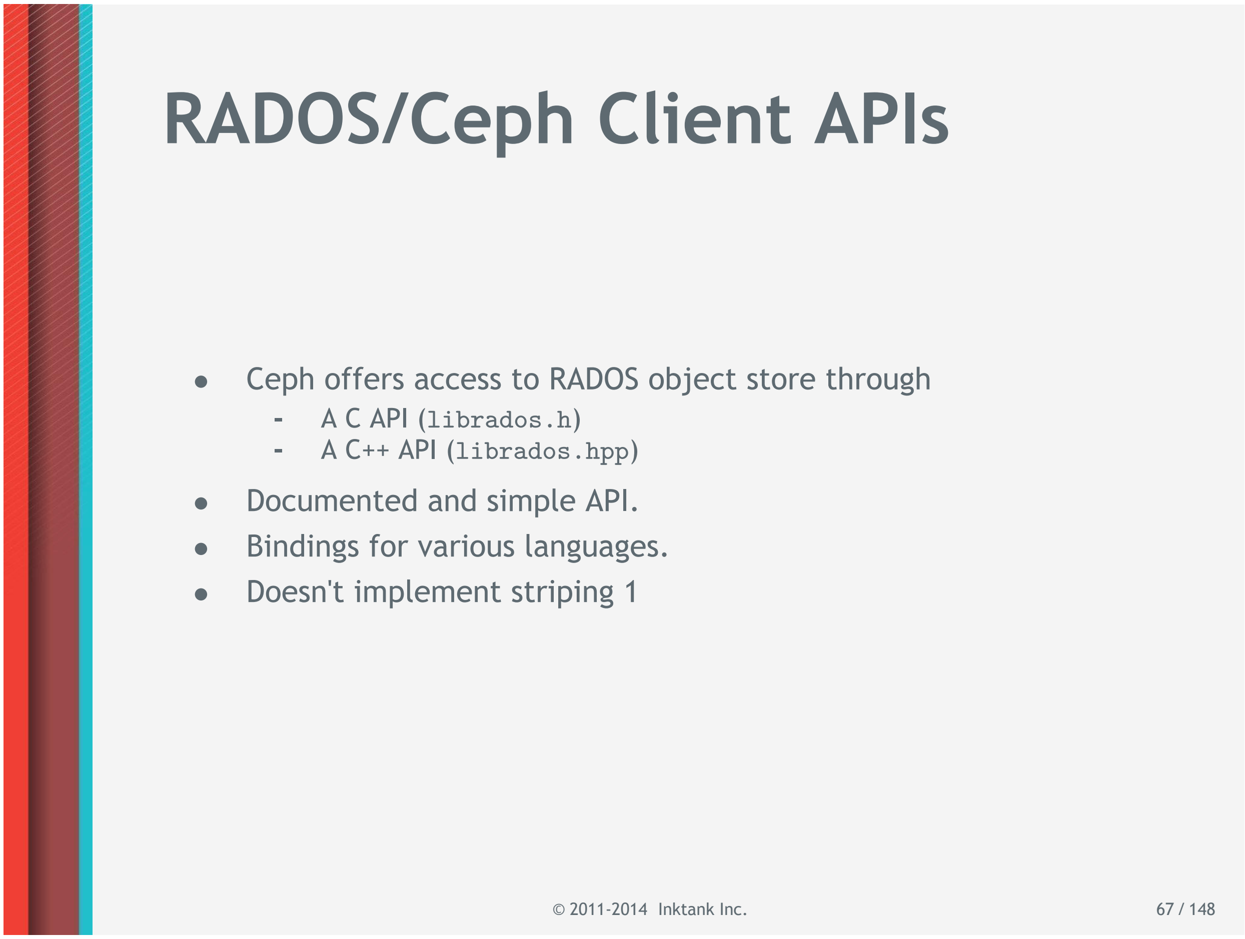
OSD Storage for RADOS objects

- On the OSD local storage
  - In any `user_xattr` capable file system.
- File system choice
  - `btrfs` will be recommended "when it's ready□
  - `xfs` is the best option for production use today
  - The contents of the file in the underlying local file system.
  - rados command-line utility is one of the many standard Ceph tools



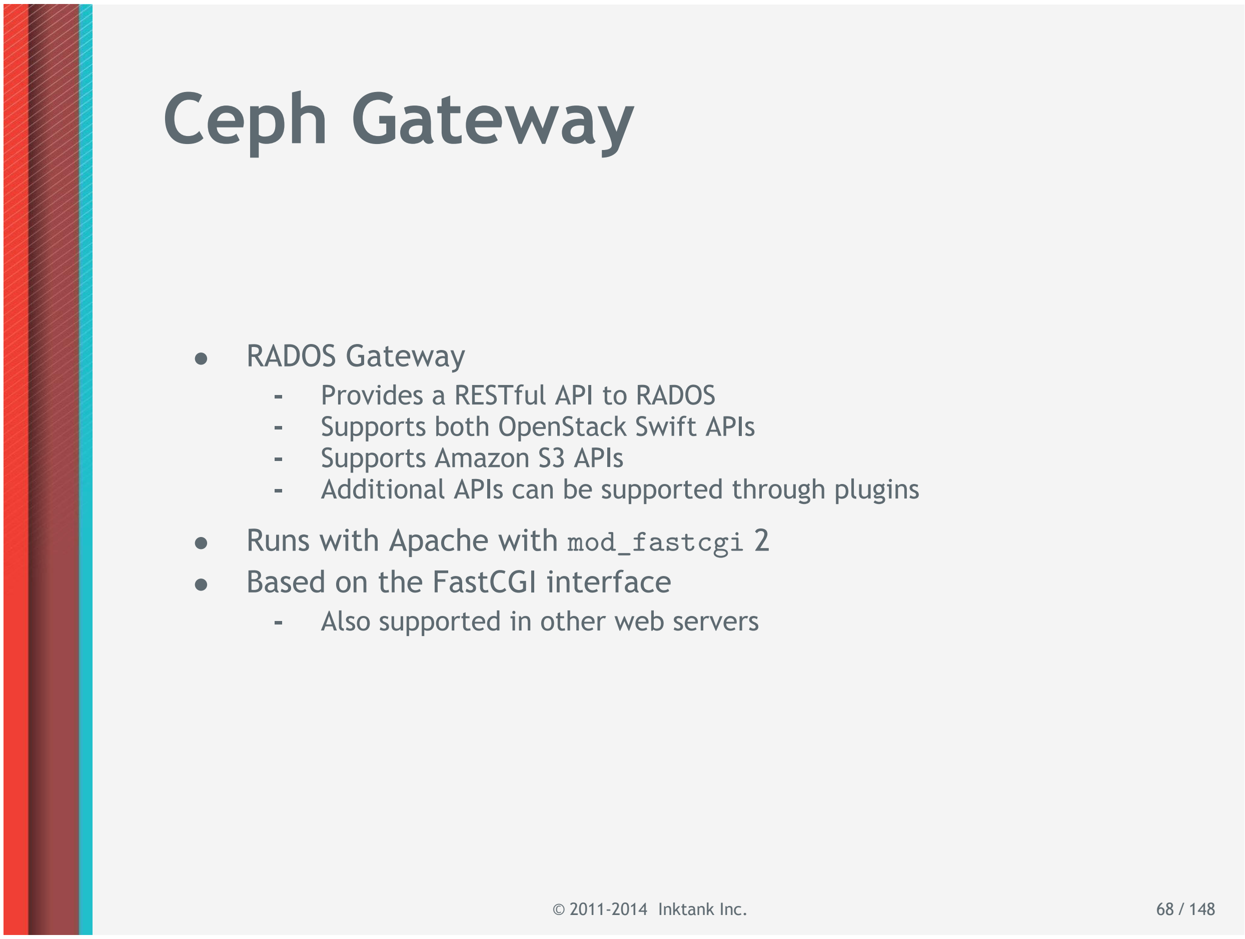
Monitors (MONs)

- Monitor Servers (MONs)
  - They arbitrate between OSDs
  - Maintain the Ceph Storage Cluster quorum
- Quorum management
  - Based on the Paxos distributed consensus algorithm
- CAP theorem of distributed systems
  - Ceph MONs pick Consistency and Partition tolerance over Availability
    - A non-quorate partition is unavailable



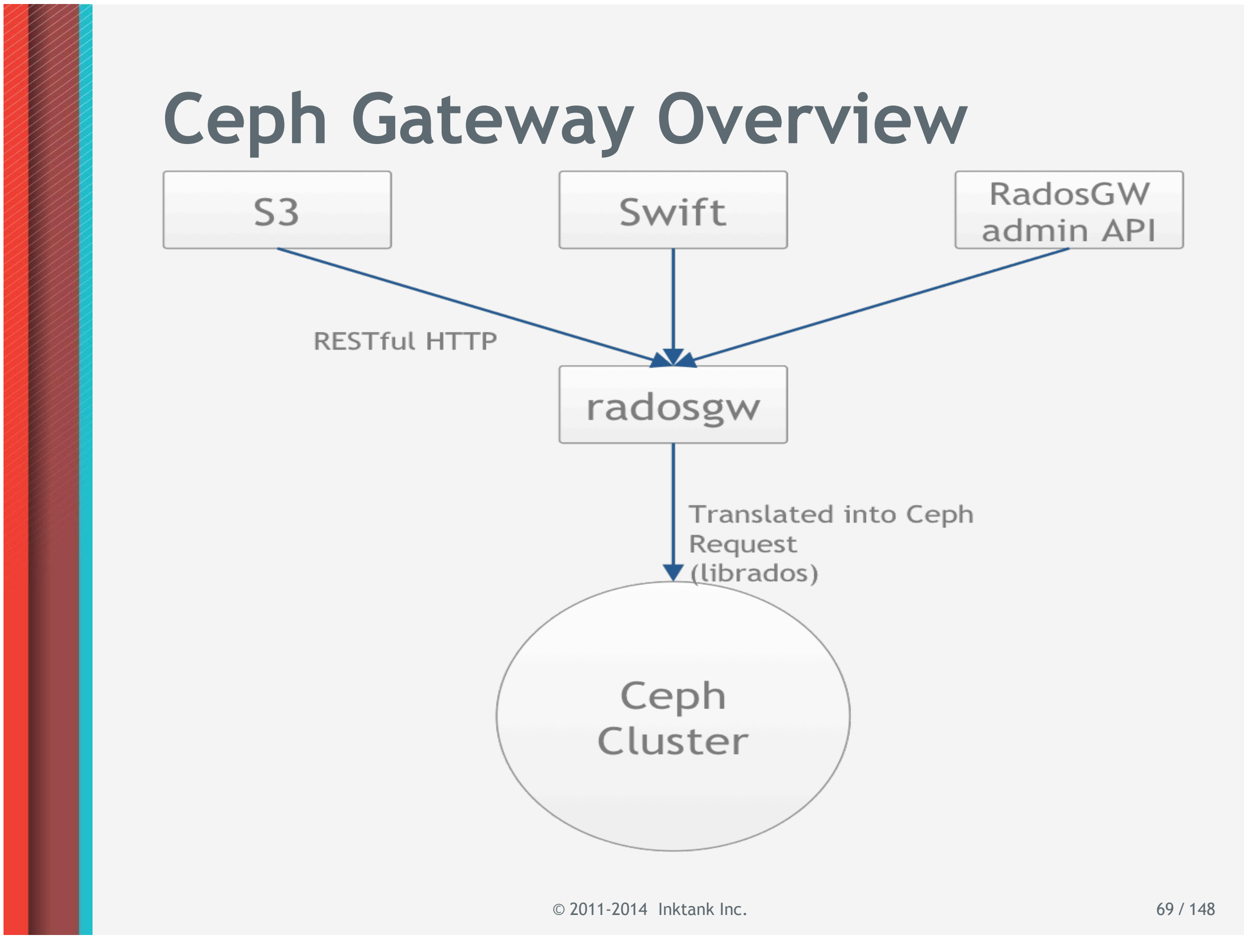
RADOS/Ceph Client APIs

- Ceph offers access to RADOS object store through
  - A C API (`librados.h`)
  - A C++ API (`librados.hpp`)
- Documented and simple API.
- Bindings for various languages.
- Doesn't implement striping 1



Ceph Gateway

- RADOS Gateway
  - Provides a RESTful API to RADOS
  - Supports both OpenStack Swift APIs
  - Supports Amazon S3 APIs
  - Additional APIs can be supported through plugins
- Runs with Apache with `mod_fastcgi` 2
- Based on the FastCGI interface
  - Also supported in other web servers



Ceph Gateway Overview

S3

Swift

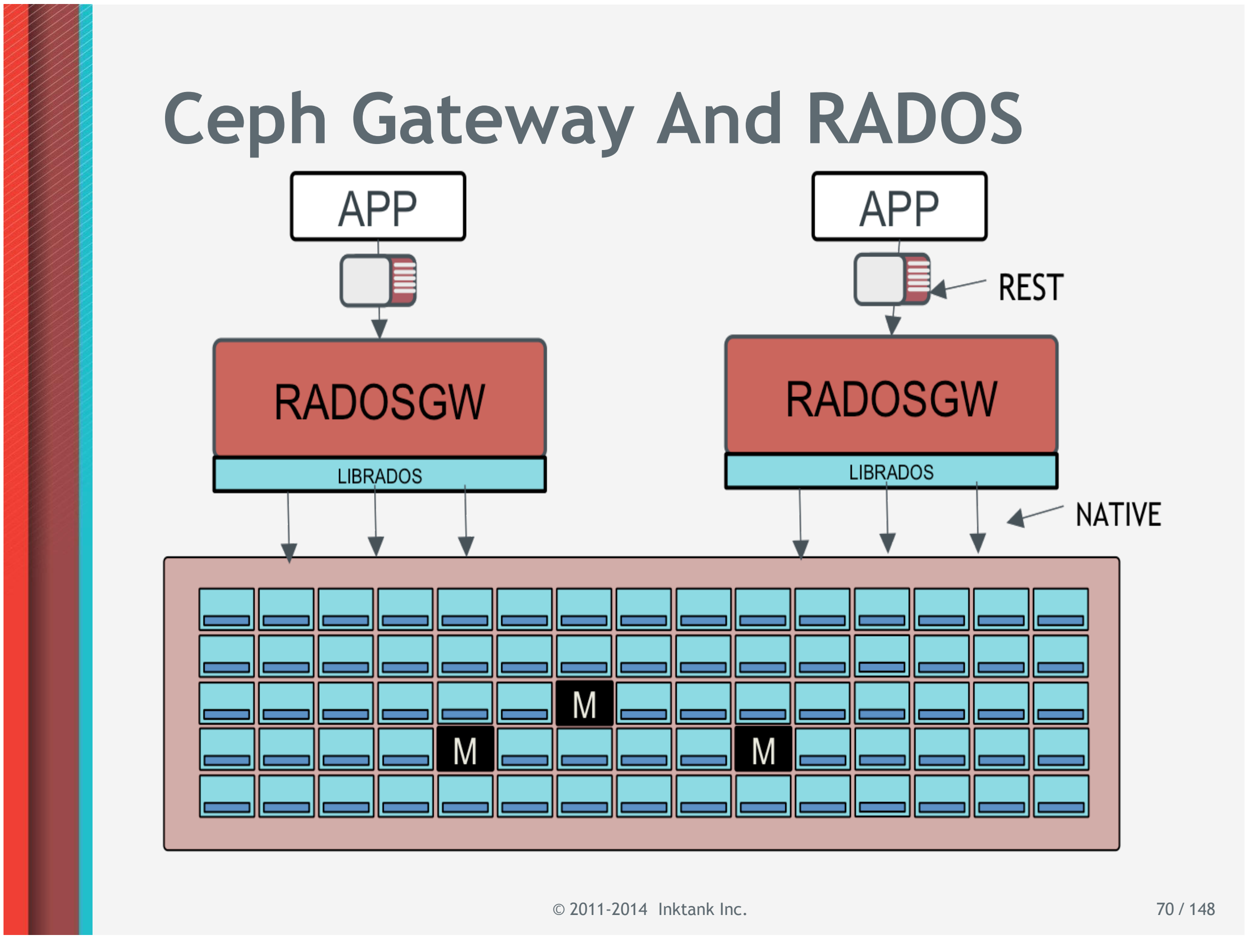
RadosGW
admin API

RESTful HTTP

radosgw

Translated into Ceph
Request
(librados)

Ceph
Cluster



Ceph Gateway And RADOS

APP

APP

REST

RADOSGW

RADOSGW

LIBRADOS

LIBRADOS

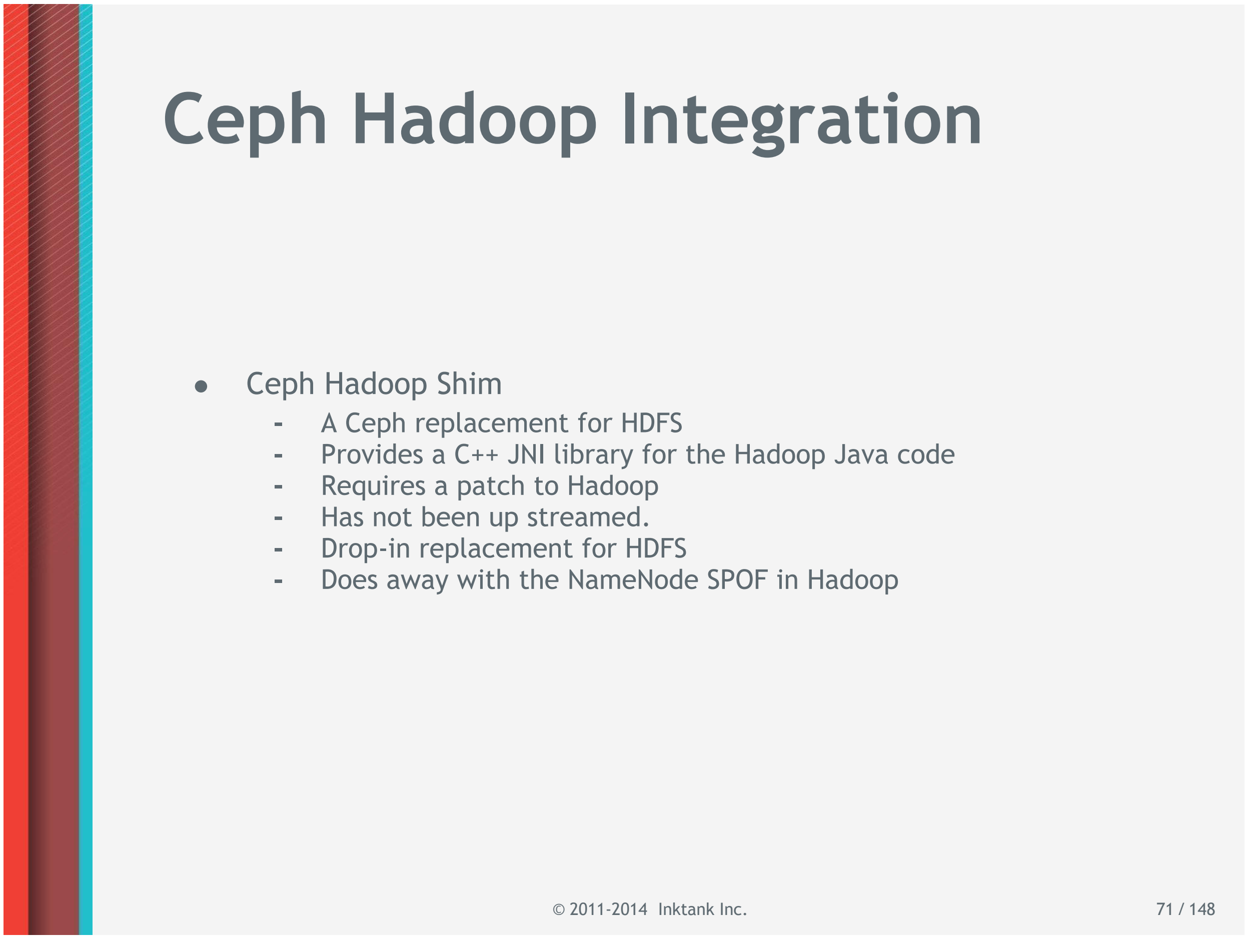
NATIVE

M

M

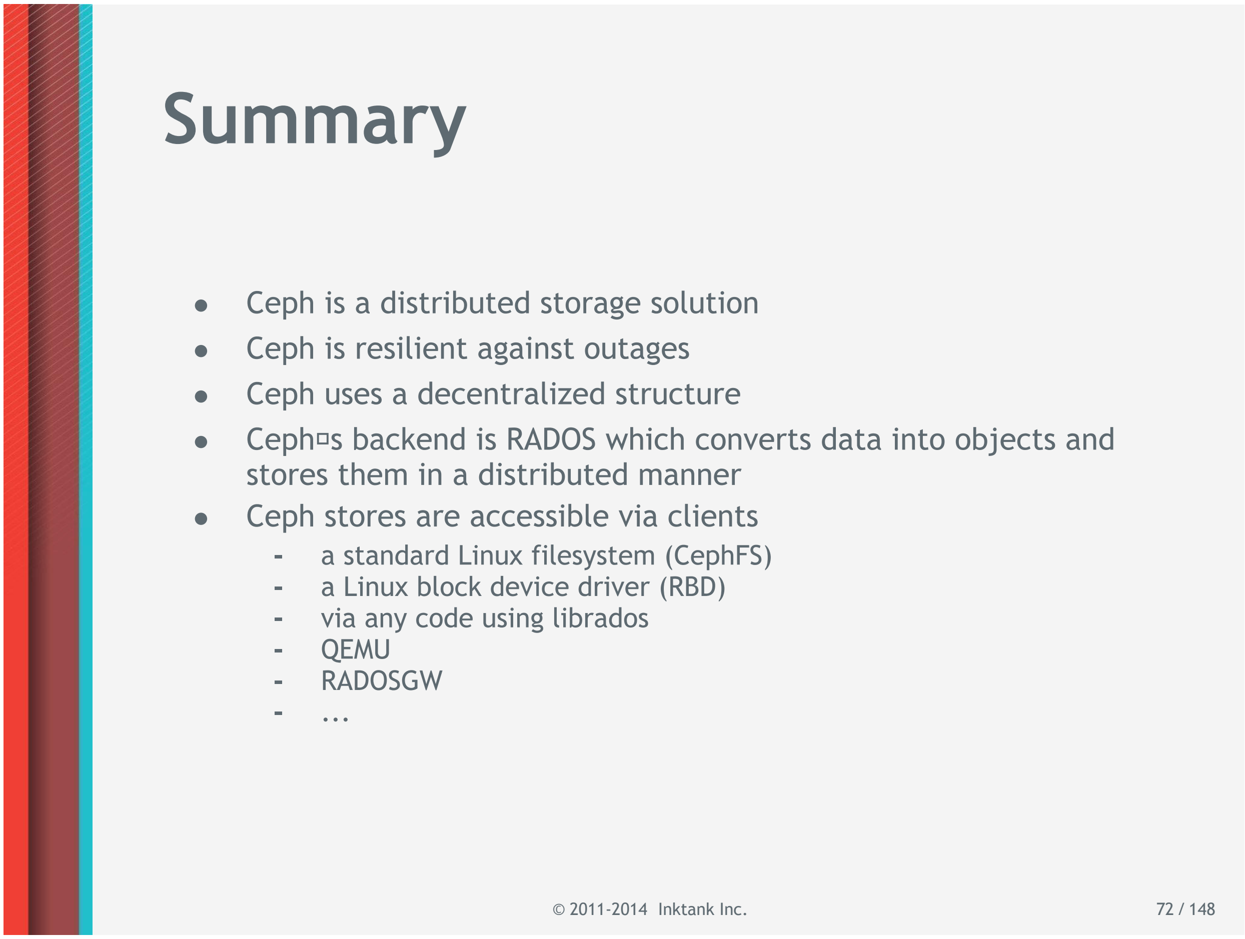
M

© 2011-2014 Inktank Inc.



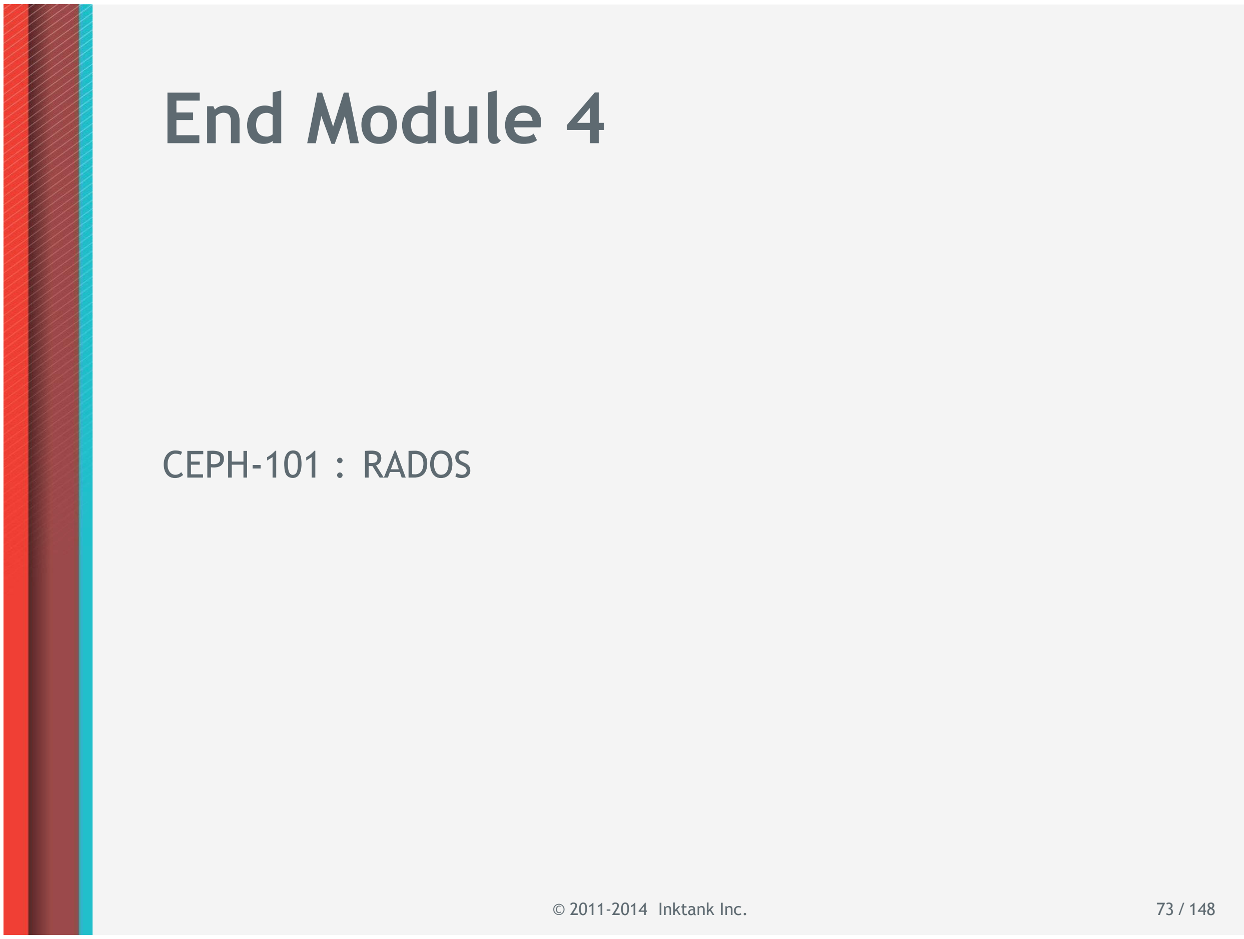
Ceph Hadoop Integration

- Ceph Hadoop Shim
  - A Ceph replacement for HDFS
  - Provides a C++ JNI library for the Hadoop Java code
  - Requires a patch to Hadoop
  - Has not been up streamed.
  - Drop-in replacement for HDFS
  - Does away with the NameNode SPOF in Hadoop



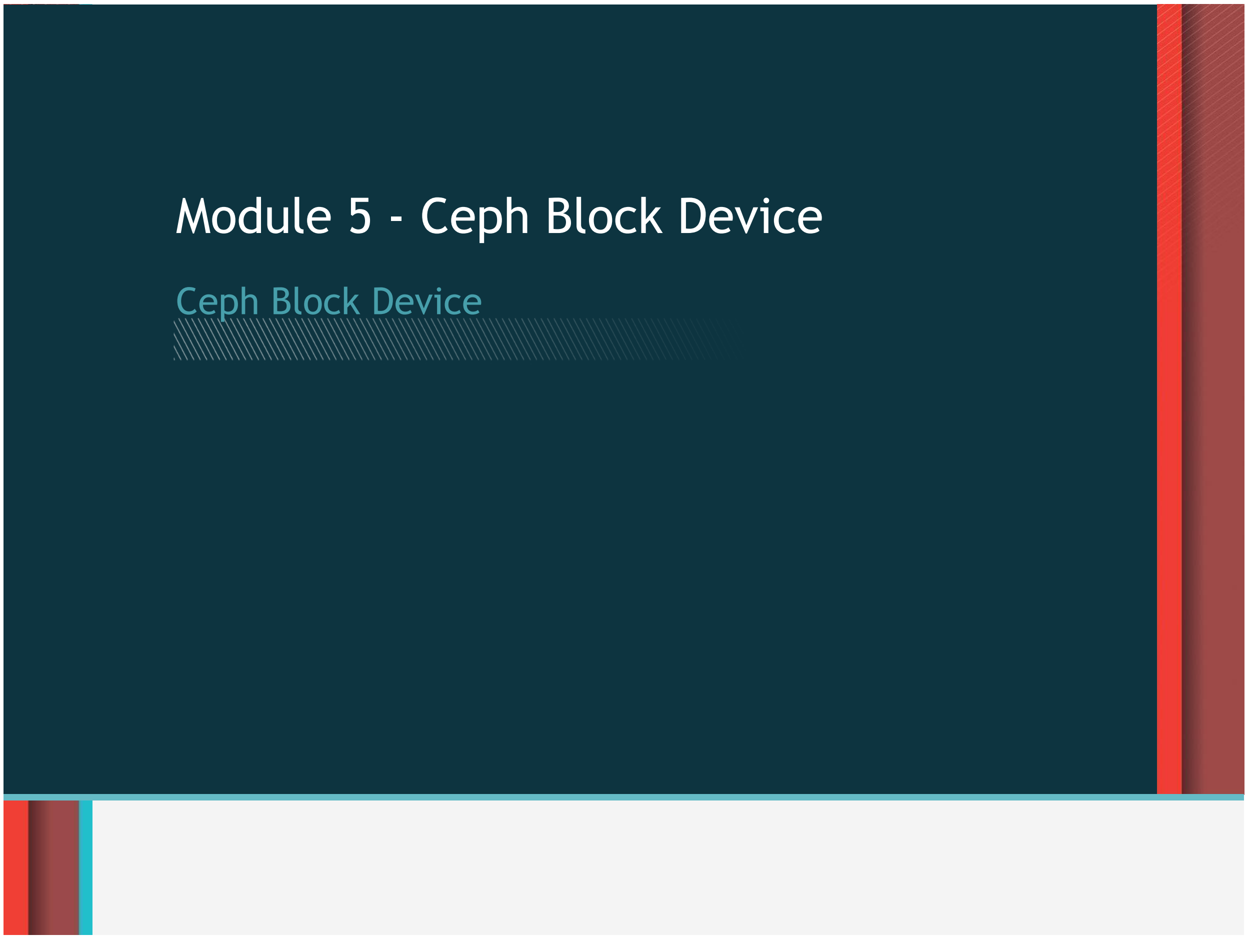
Summary

- Ceph is a distributed storage solution
- Ceph is resilient against outages
- Ceph uses a decentralized structure
- Ceph□s backend is RADOS which converts data into objects and stores them in a distributed manner
- Ceph stores are accessible via clients
  - a standard Linux filesystem (CephFS)
  - a Linux block device driver (RBD)
  - via any code using librados
  - QEMU
  - RADOSGW
  - ...



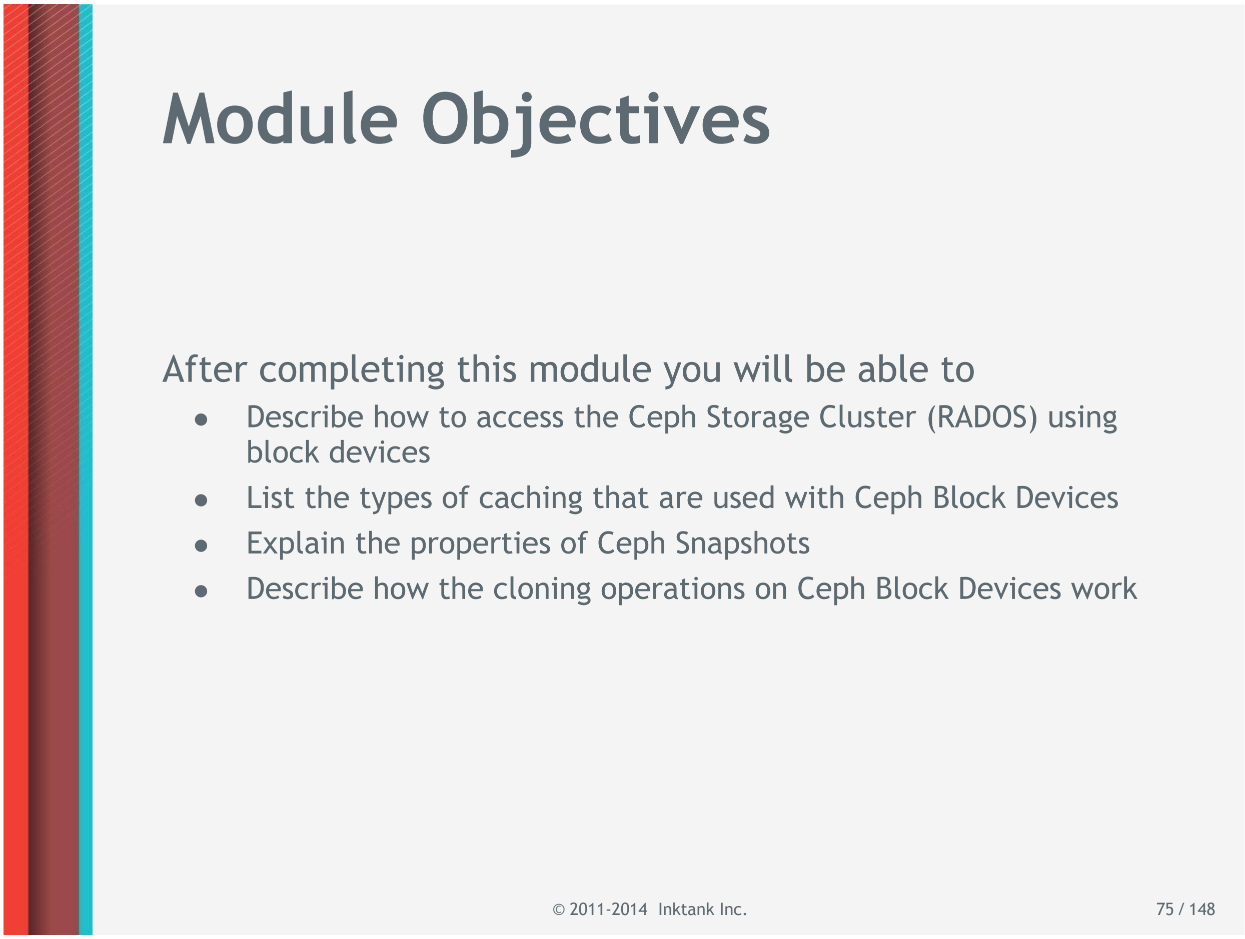
End Module 4

CEPH-101 : RADOS



Module 5 - Ceph Block Device

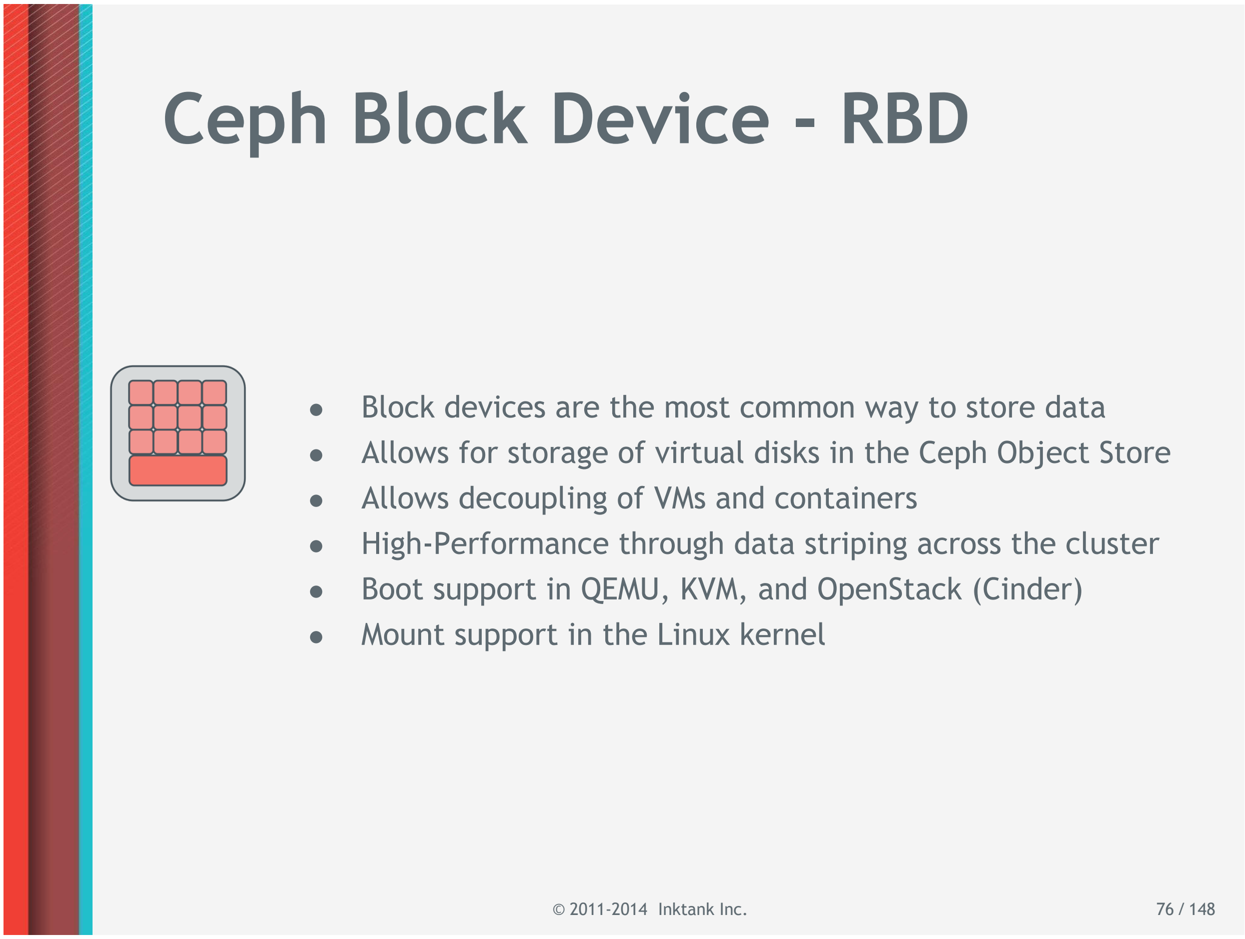
Ceph Block Device



Module Objectives

After completing this module you will be able to

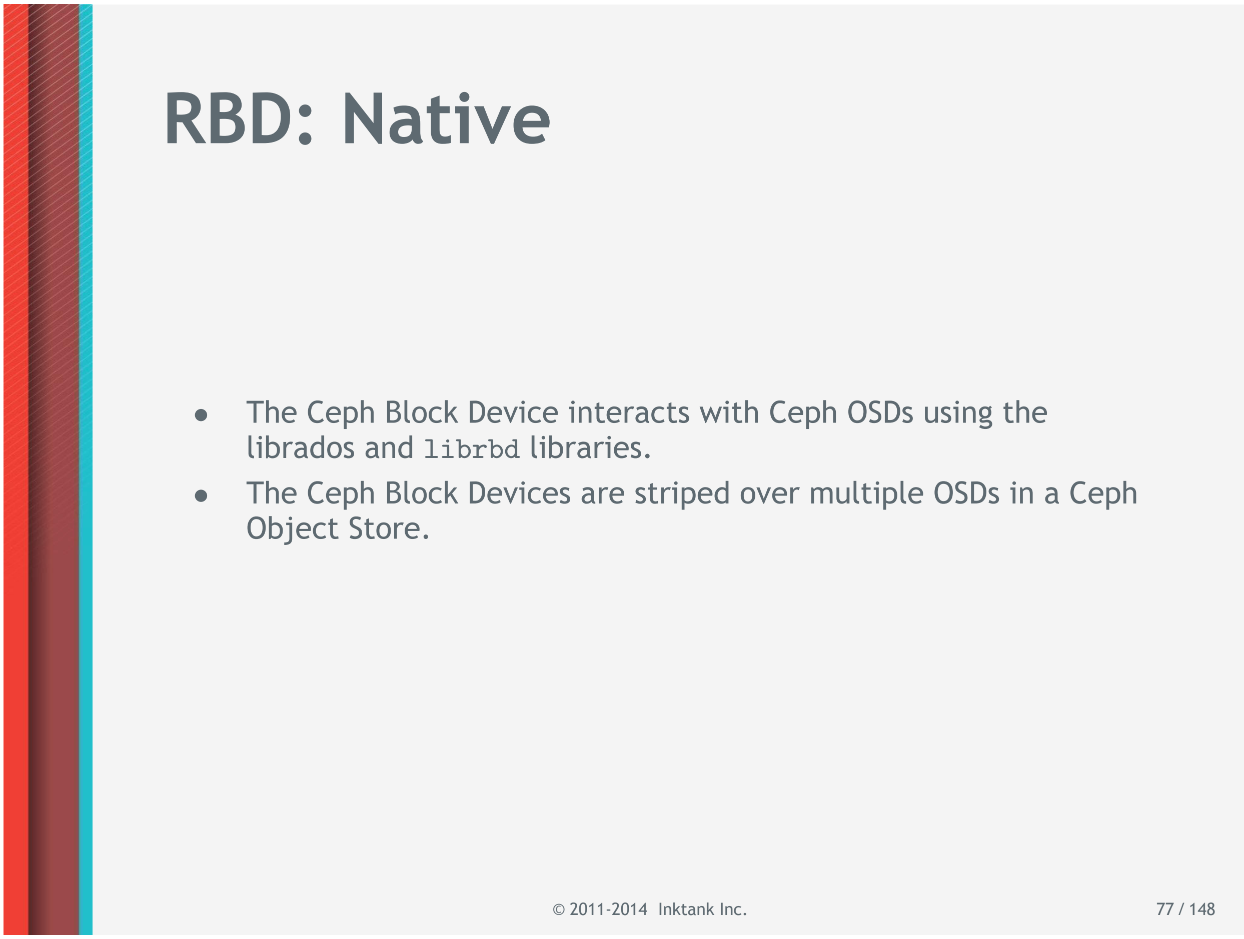
- Describe how to access the Ceph Storage Cluster (RADOS) using block devices
- List the types of caching that are used with Ceph Block Devices
- Explain the properties of Ceph Snapshots
- Describe how the cloning operations on Ceph Block Devices work



Ceph Block Device - RBD

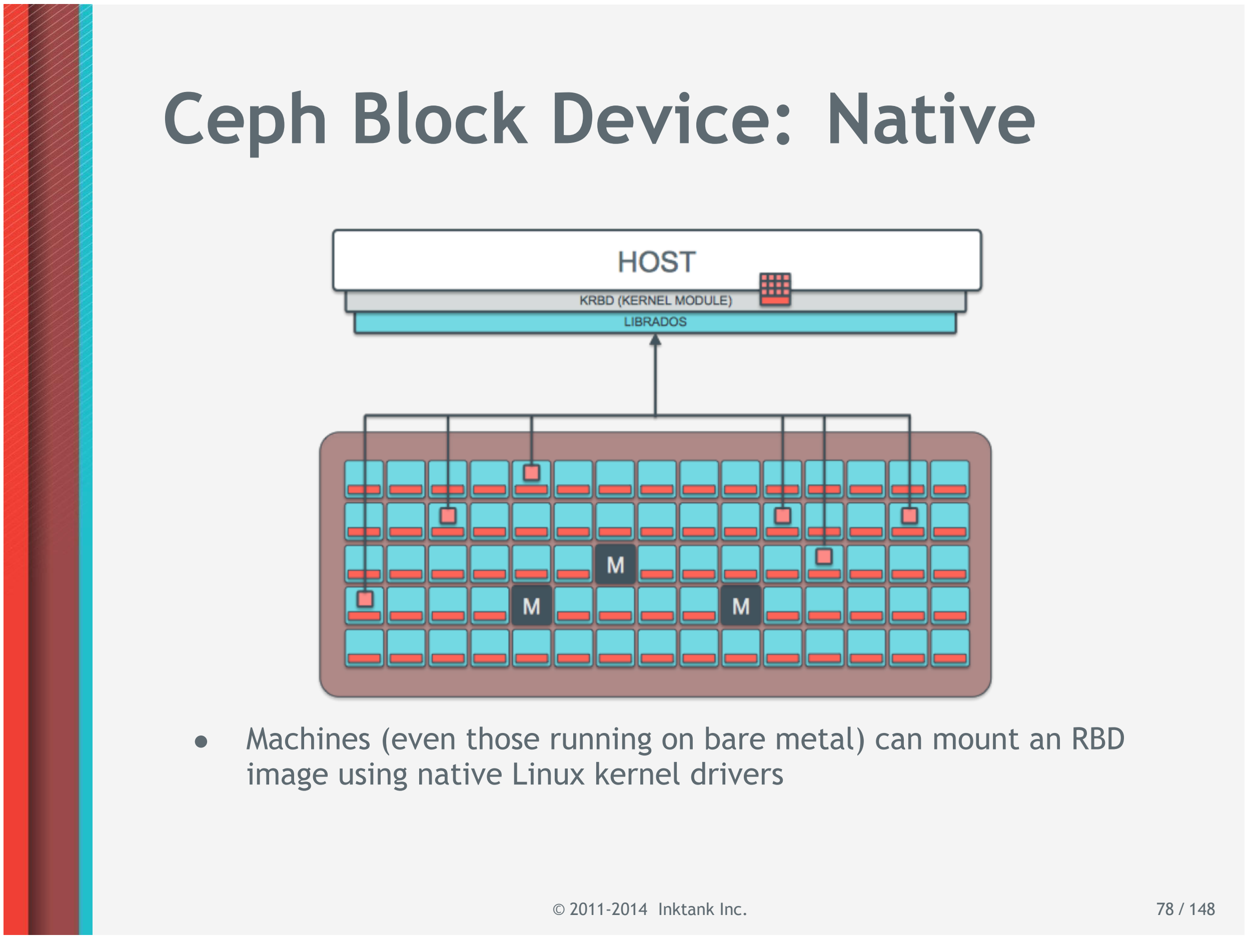
- Block devices are the most common way to store data
- Allows for storage of virtual disks in the Ceph Object Store
- Allows decoupling of VMs and containers
- High-Performance through data striping across the cluster
- Boot support in QEMU, KVM, and OpenStack (Cinder)
- Mount support in the Linux kernel

RBD stands for RADOS Block Device
Date is striped across the Ceph cluster



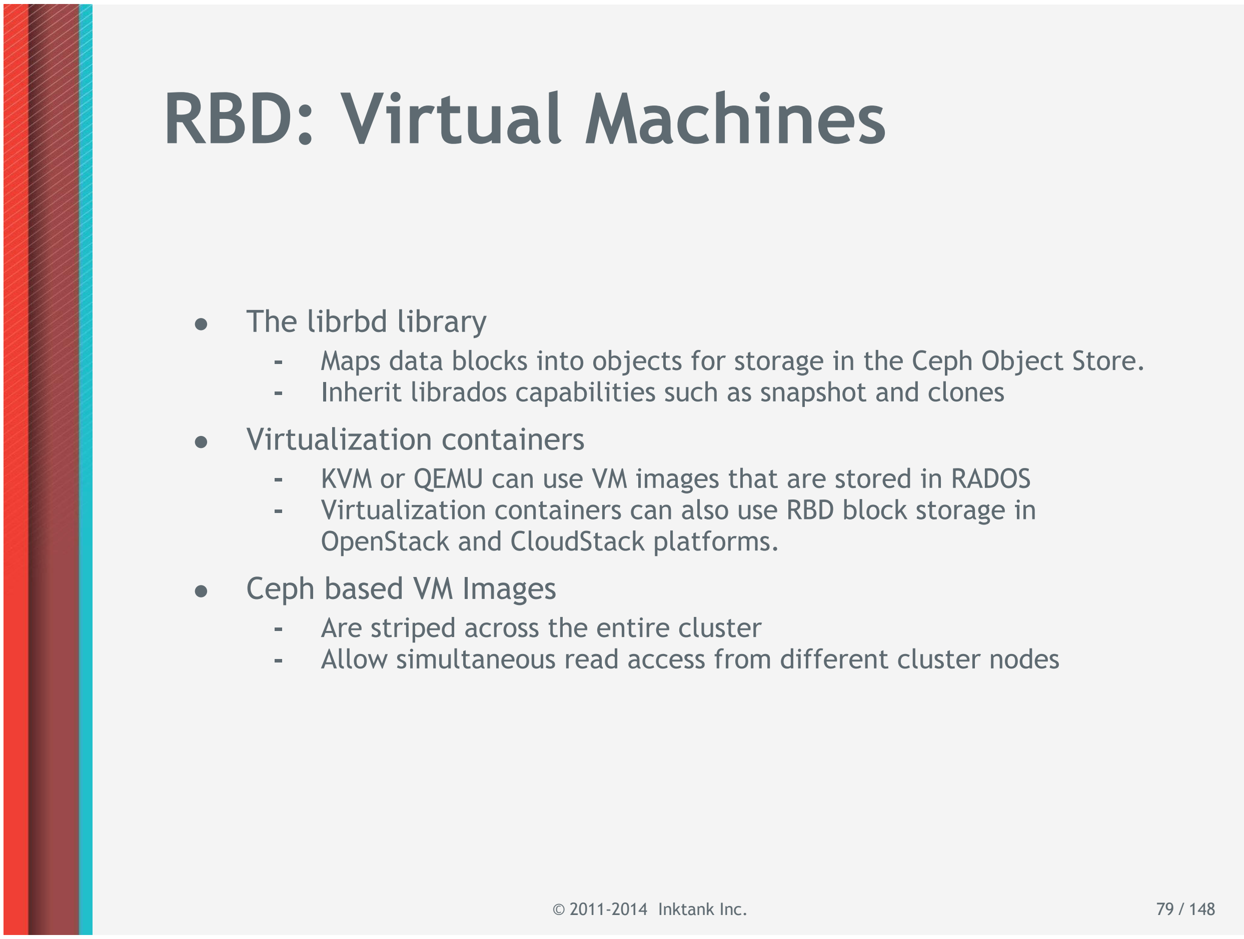
RBD: Native

- The Ceph Block Device interacts with Ceph OSDs using the librados and `librbd` libraries.
- The Ceph Block Devices are striped over multiple OSDs in a Ceph Object Store.



Ceph Block Device: Native

- Machines (even those running on bare metal) can mount an RBD image using native Linux kernel drivers

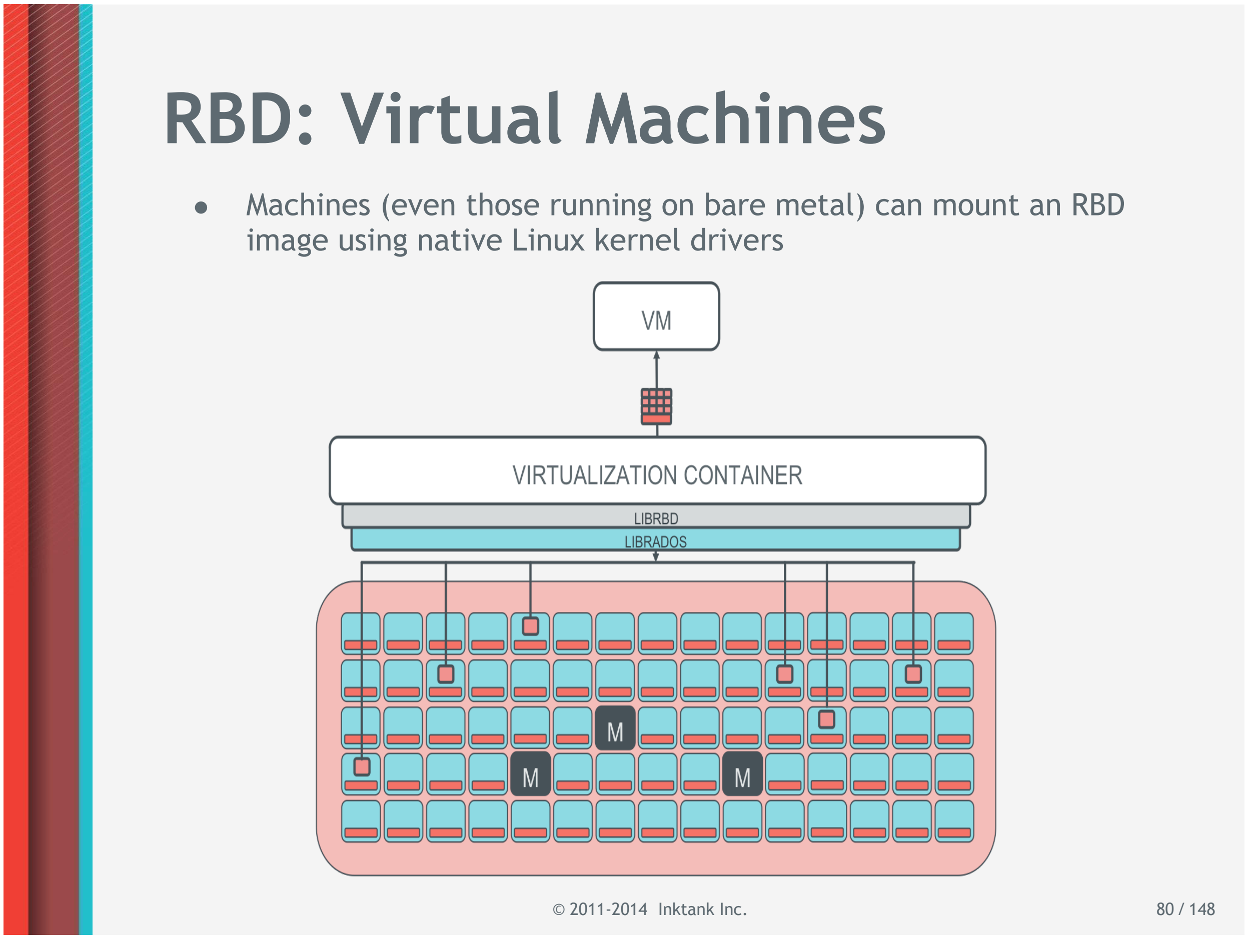


RBD: Virtual Machines

- The librbd library
  - Maps data blocks into objects for storage in the Ceph Object Store.
  - Inherit librados capabilities such as snapshot and clones
- Virtualization containers
  - KVM or QEMU can use VM images that are stored in RADOS
  - Virtualization containers can also use RBD block storage in OpenStack and CloudStack platforms.
- Ceph based VM Images
  - Are striped across the entire cluster
  - Allow simultaneous read access from different cluster nodes

Virtualization containers can boot a VM without transferring the boot image to the VM itself.

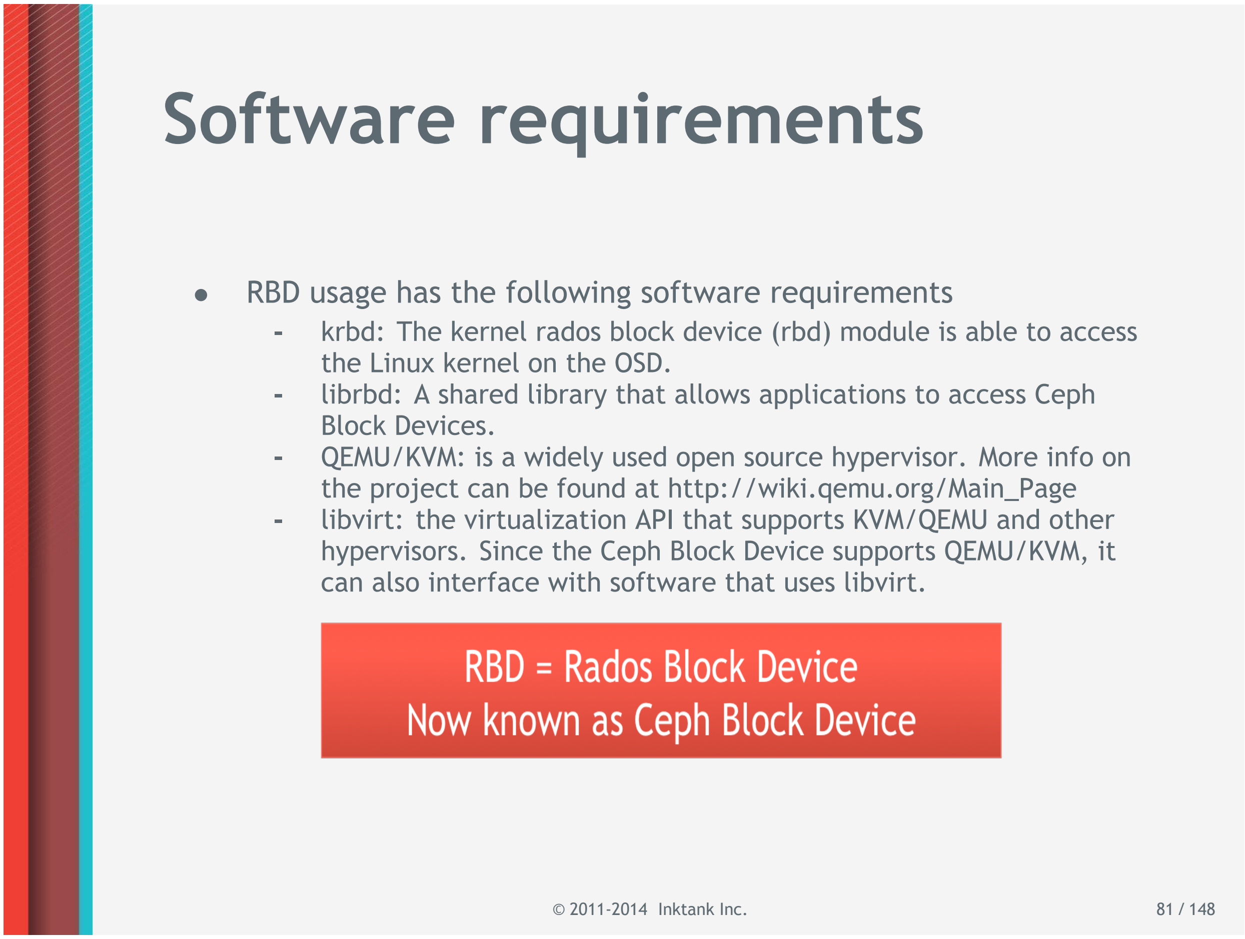
Config file rbdmap will tell which RBD device needs to be mounted



RBD: Virtual Machines

- Machines (even those running on bare metal) can mount an RBD image using native Linux kernel drivers

As far as the VM is concerned, it sees a block device and is not even aware about the CEPH cluster.



Software requirements

- RBD usage has the following software requirements
  - krbd: The kernel rados block device (rbd) module is able to access the Linux kernel on the OSD.
  - librbd: A shared library that allows applications to access Ceph Block Devices.
  - QEMU/KVM: is a widely used open source hypervisor. More info on the project can be found at http://wiki.qemu.org/Main_Page
  - libvirt: the virtualization API that supports KVM/QEMU and other hypervisors. Since the Ceph Block Device supports QEMU/KVM, it can also interface with software that uses libvirt.

RBD = Rados Block Device
Now known as Ceph Block Device

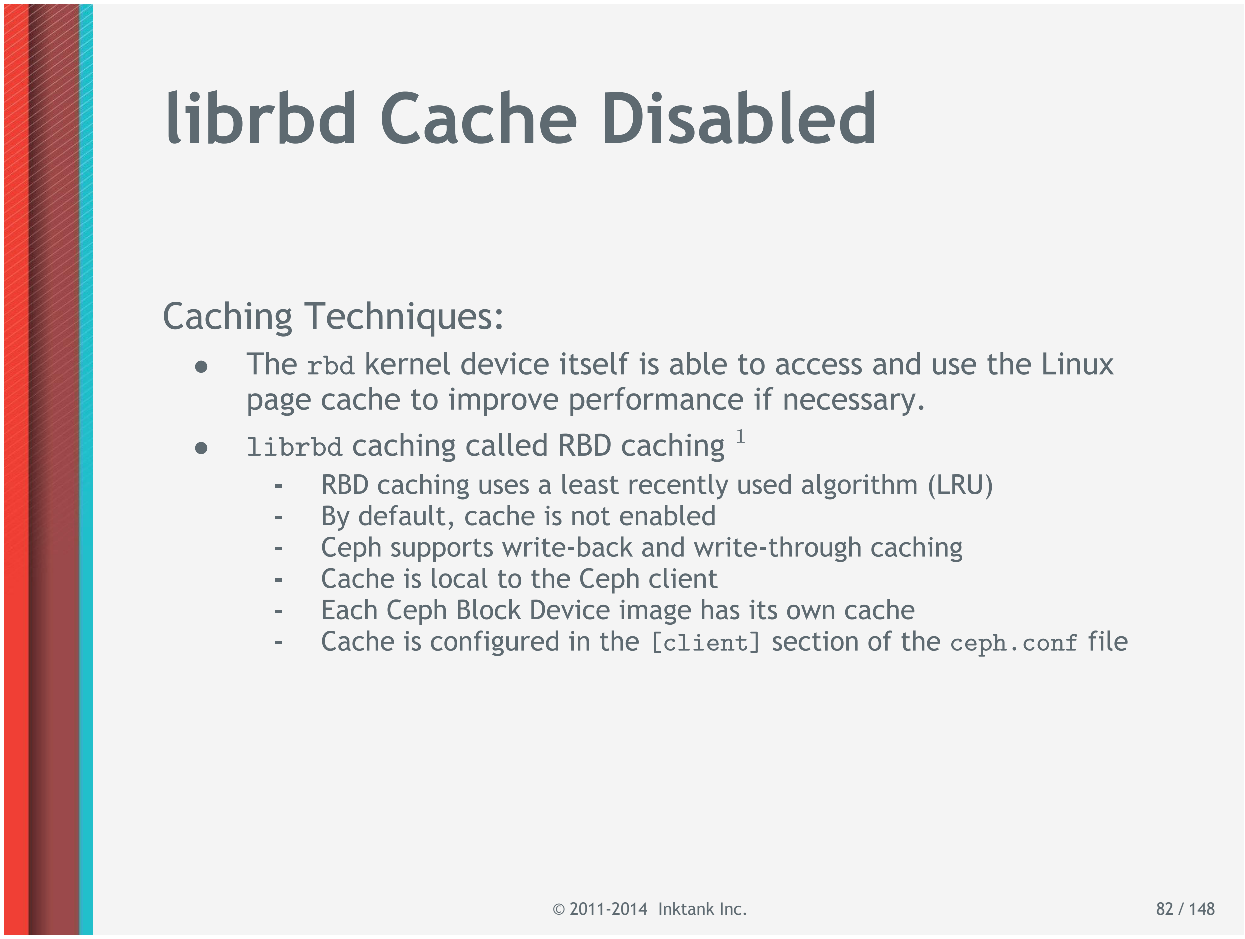
You will be dependant on the kernel version for the best performance and avoiding the bugs

A kernel version of minimum 3.8 is SUPER highly recommended

To use an RBD directly in the VM itself, you need to:

Install librbd (will also install librados)

Then map the RBD device



librbd Cache Disabled

Caching Techniques:

- The `rbd` kernel device itself is able to access and use the Linux page cache to improve performance if necessary.
- `librbd` caching called RBD caching [1]
  - RBD caching uses a least recently used algorithm (LRU)
  - By default, cache is not enabled
  - Ceph supports write-back and write-through caching
  - Cache is local to the Ceph client
  - Each Ceph Block Device image has its own cache
  - Cache is configured in the `[client]` section of the `ceph.conf` file

LIBRBD can not leverage the Linux page caching for its own use. Therefore LIBRBD implements its own caching mechanism

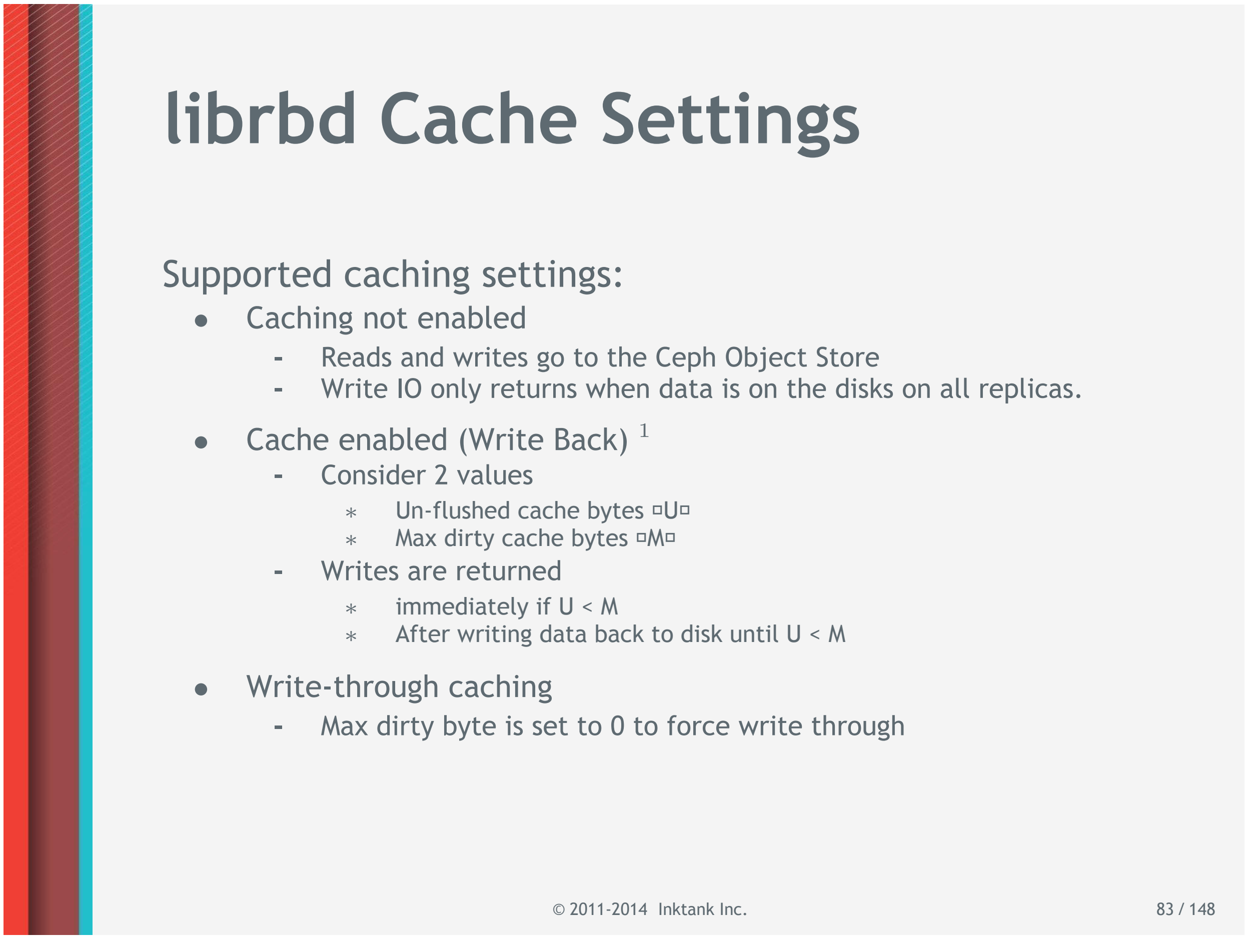
By default caching is disabled in LIBRBD.

Note 1 : In write-back mode LIBRBD caching can coalesce contiguous requests for better throughput.

We offer Write Back (aka Cache Enabled which is the activation default) and Write Through support

Be cautious with Write Back as the host will be caching and acknowledge the write IO request as soon as data is place in the server LIBRBD local cache.

Write Through is highly recommended for production servers to avoid loosing data in case of a server failure



librbd Cache Settings

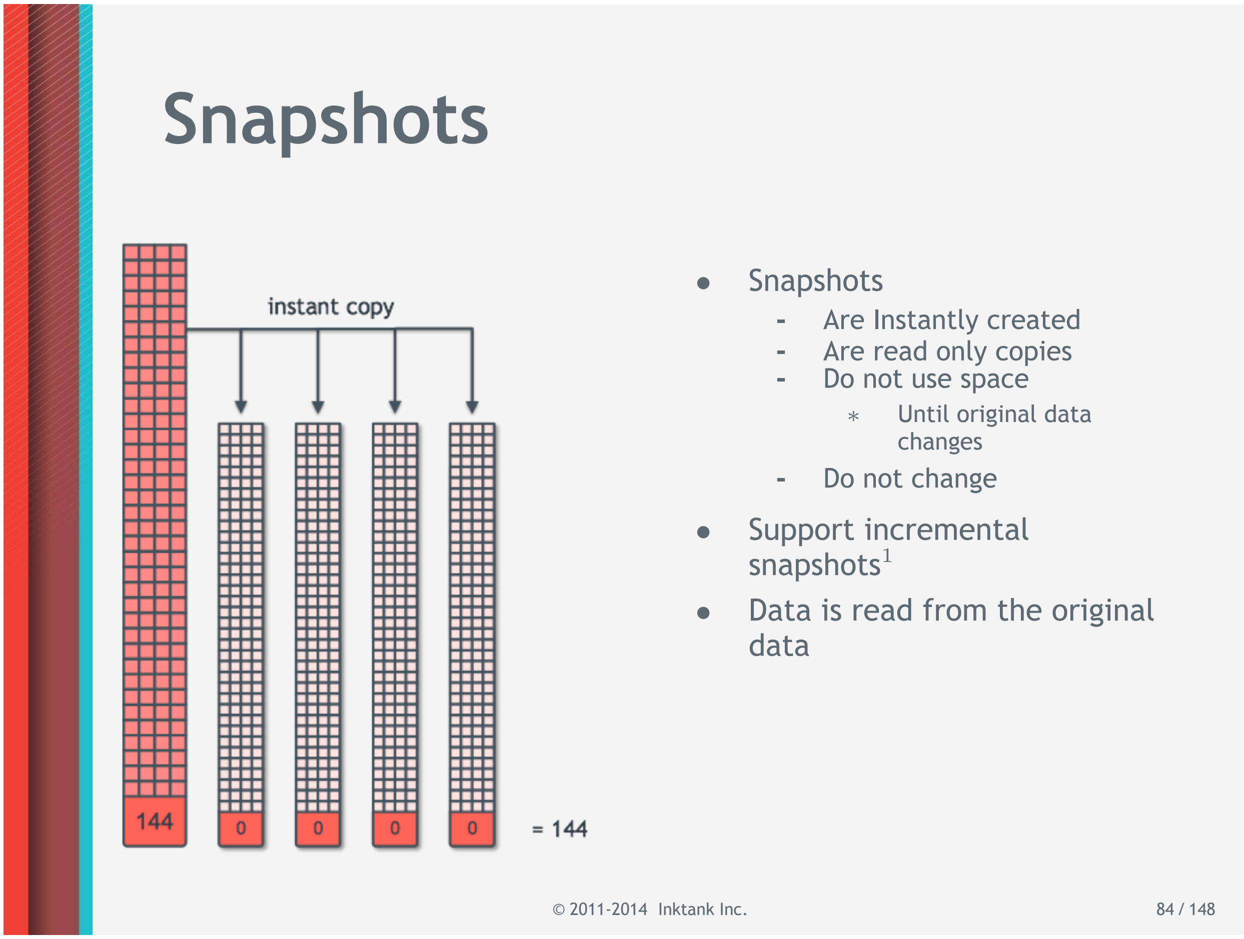
Supported caching settings:

- Caching not enabled
  - Reads and writes go to the Ceph Object Store
  - Write IO only returns when data is on the disks on all replicas.
- Cache enabled (Write Back) [1]
  - Consider 2 values
    - Un-flushed cache bytes ¤U¤
    - Max dirty cache bytes ¤M¤
  - Writes are returned
    - immediately if U < M
    - After writing data back to disk until U < M
- Write-through caching
  - Max dirty byte is set to 0 to force write through

Note 1: In write-back mode it can coalesce contiguous requests for better throughput.

The ceph.conf file settings for RBD should be set in the [client] section of your configuration file.

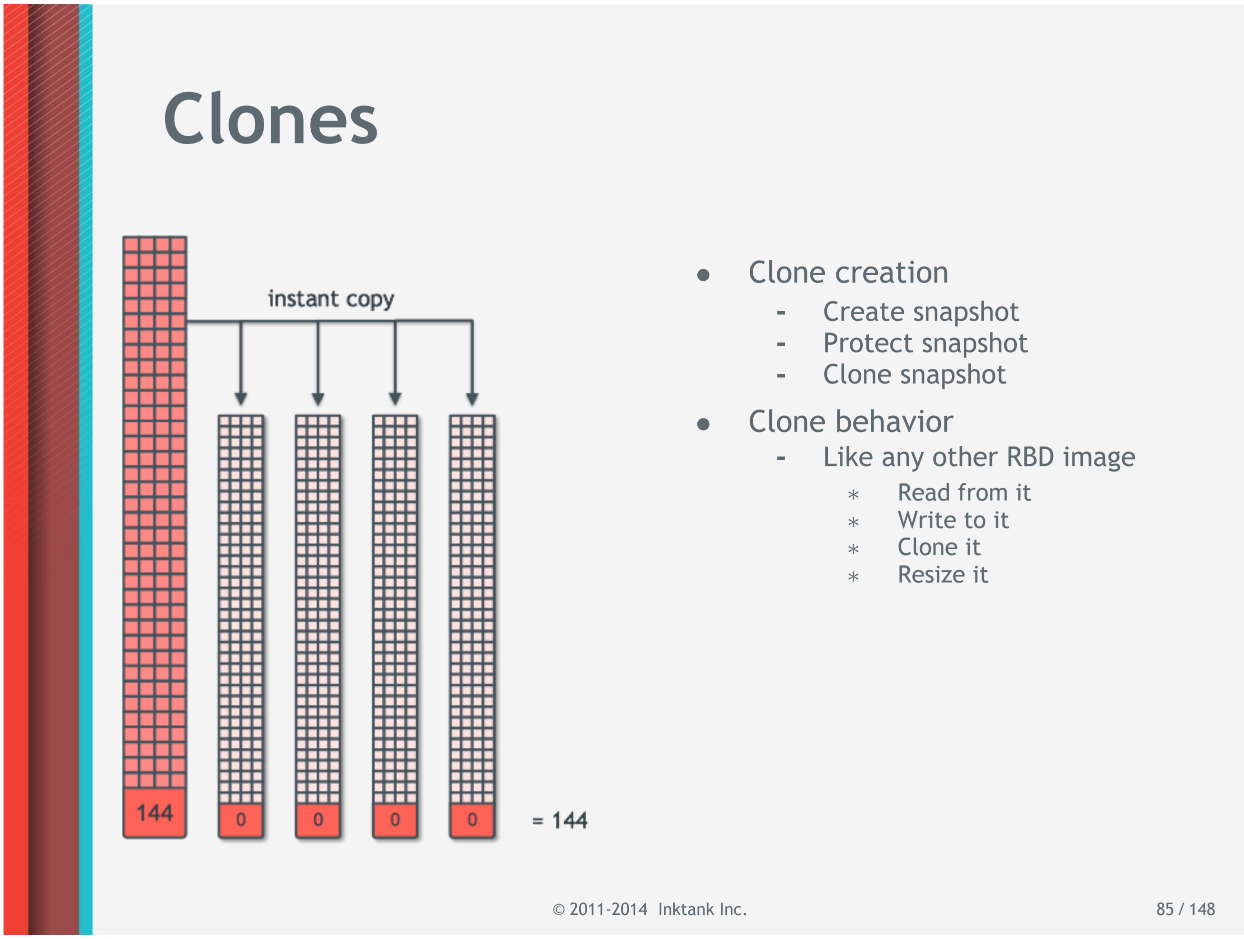
The settings include (default values are in bold underlined):
`rbd cache` Enable RBD caching. Value is true or false
`rbd cache size` The RBD cache size in bytes. Integer 32MB
`rbd cache max dirty` The dirty byte threshold that triggers write back. Must be less than above. 24MB
`rbd cache target dirty` The dirty target before cache starts writing back data . Does not hold write IOs to cache. 16MB
`rbd cache max dirty age` Number of seconds dirty bytes are kept in cache before writing back. 1
`rbd cache writethrough until flush` Start in write through mode and switch to write back after first flush occurs. Value is true or false.



Snapshots

- Snapshots
  - Are Instantly created
  - Are read only copies
  - Do not use space
    - Until original data changes
  - Do not change
- Support incremental snapshots[1]
- Data is read from the original data

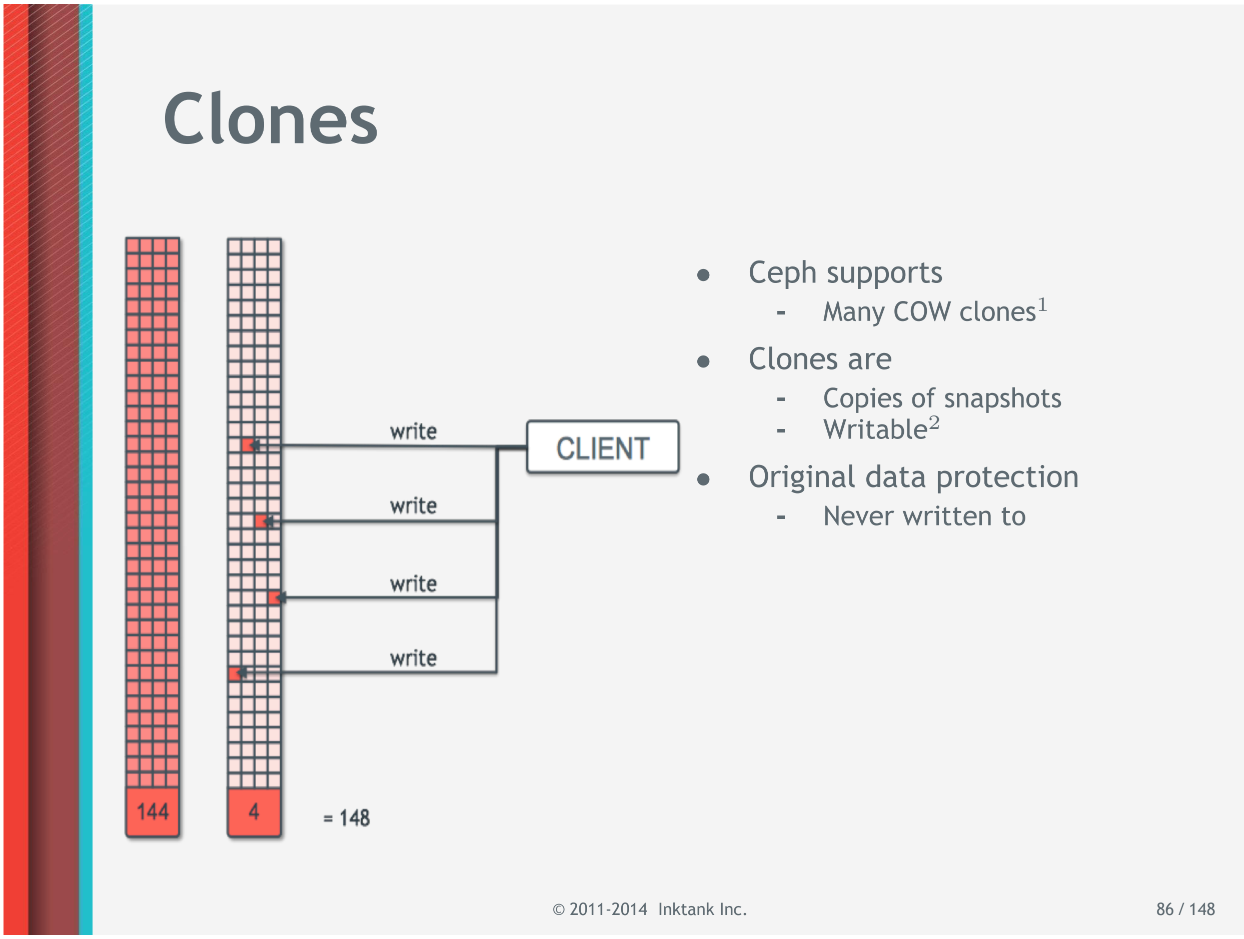
Since Cuttlefish, we support incremental snapshots



Clones

- Clone creation
  - Create snapshot
  - Protect snapshot
  - Clone snapshot
- Clone behavior
  - Like any other RBD image
    - Read from it
    - Write to it
    - Clone it
    - Resize it

A clone is created from a snapshot

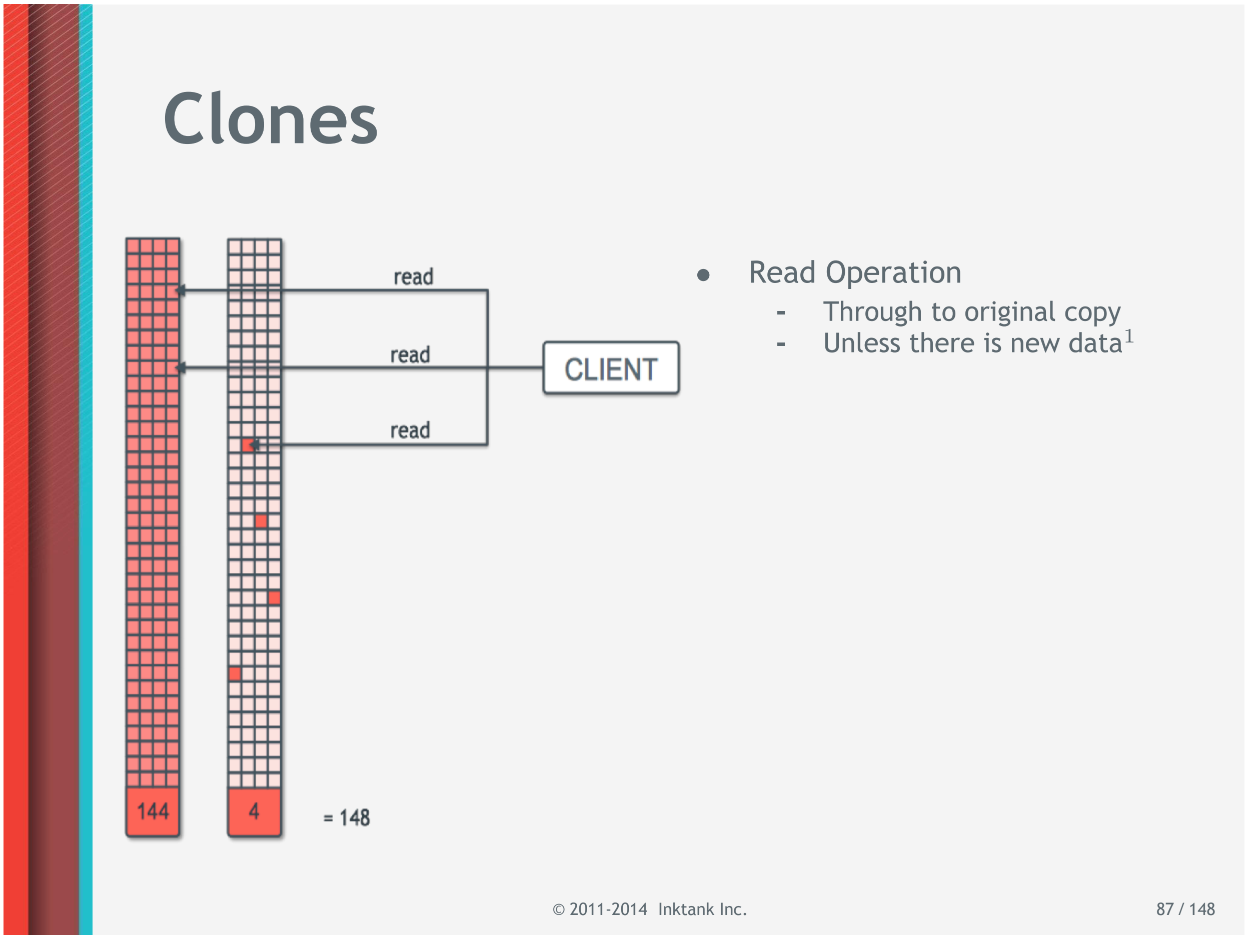


Clones

- Ceph supports
  - Many COW clones[1]
- Clones are
  - Copies of snapshots
  - Writable[2]
- Original data protection
  - Never written to

Note 1 : Reads are always served from the original snapshot used to create the clone. Ceph supports many copy-on-write clones of a snapshot

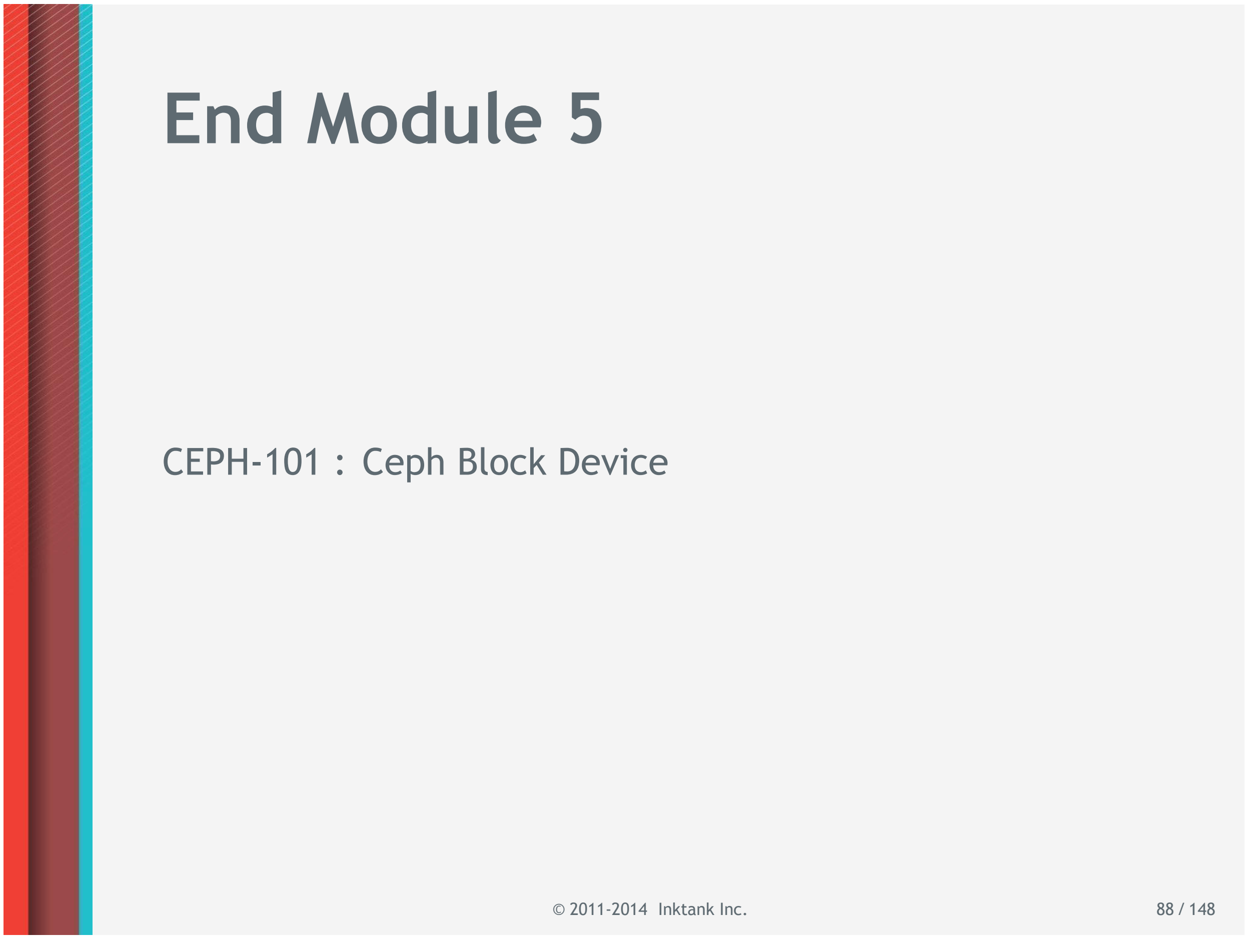
Note 2 : Snapshots are read-only!



Clones

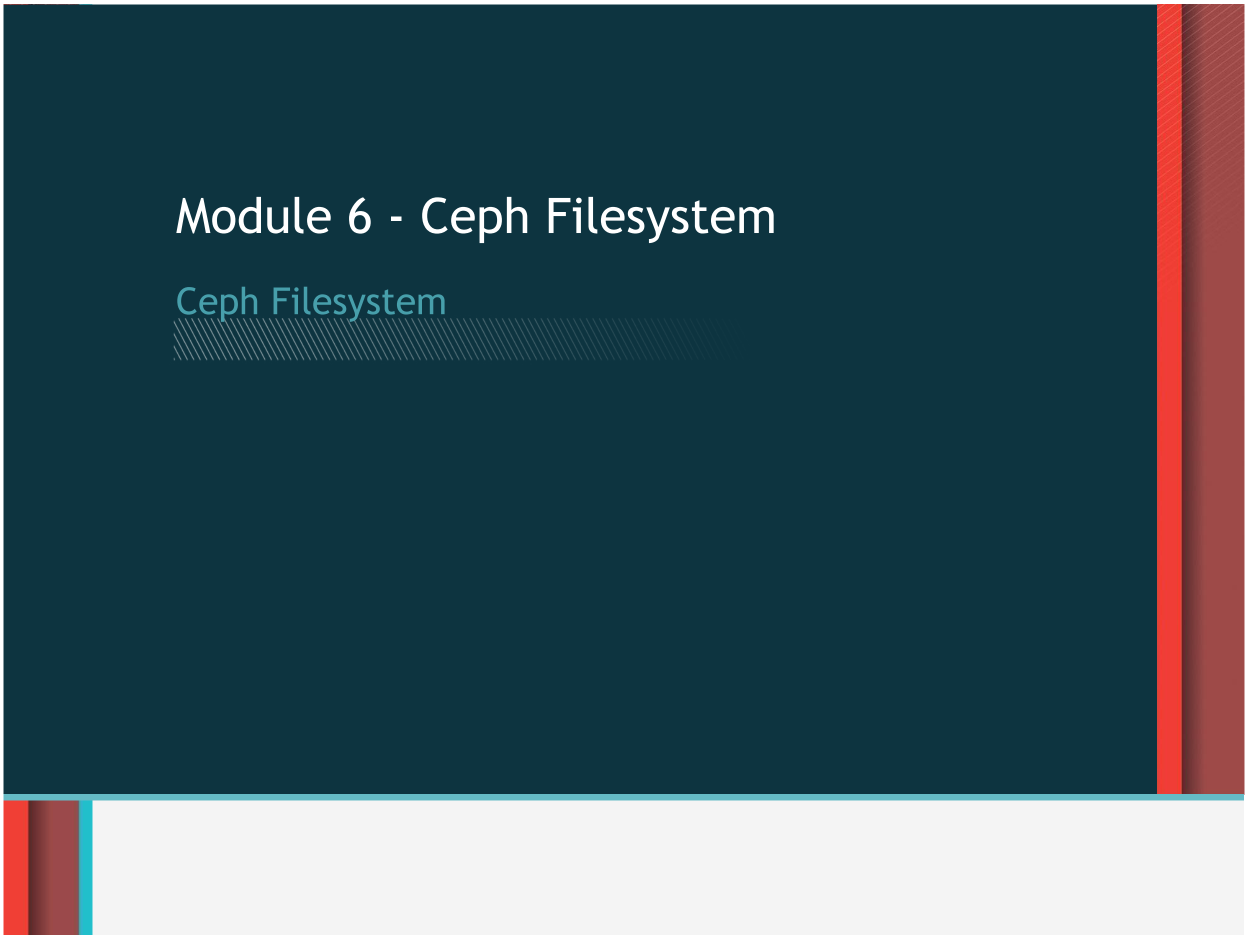
- Read Operation
  - Through to original copy
  - Unless there is new data[1]

Note 1 : If data has been updated in the clone, data is read from the clone mounted on the host.



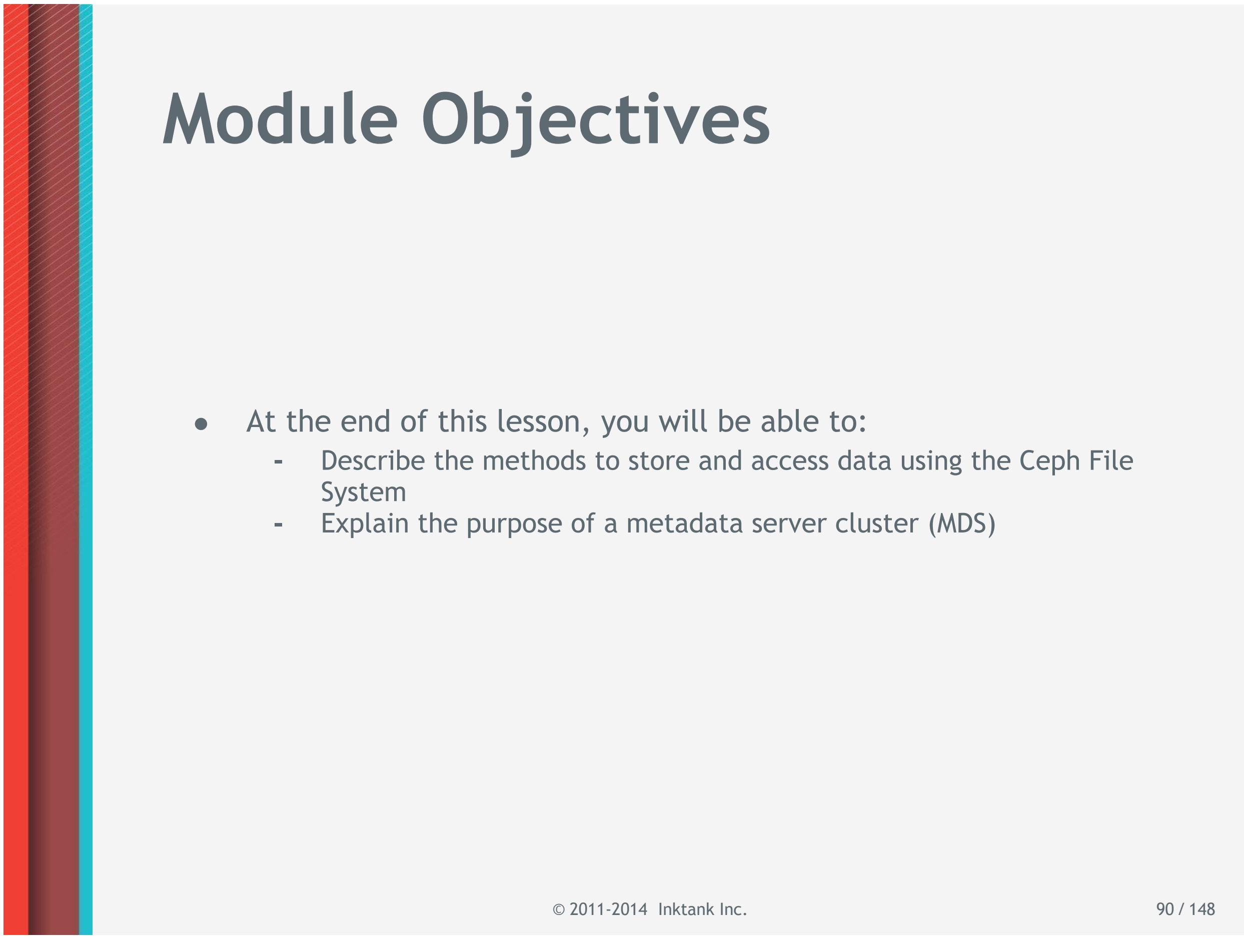
End Module 5

CEPH-101 : Ceph Block Device



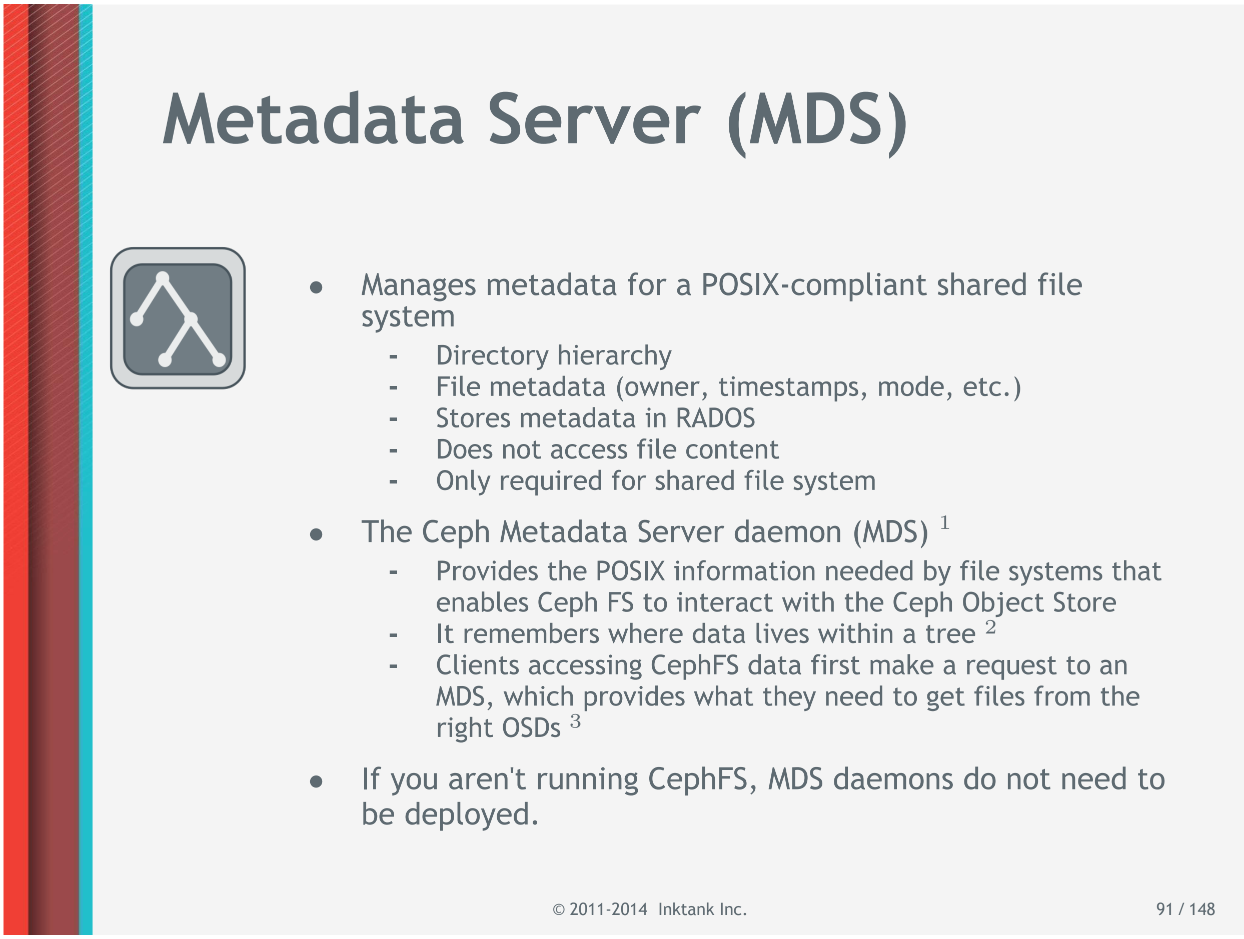
Module 6 - Ceph Filesystem

Ceph Filesystem



Module Objectives

- At the end of this lesson, you will be able to:
  - Describe the methods to store and access data using the Ceph File System
  - Explain the purpose of a metadata server cluster (MDS)



Metadata Server (MDS)

- Manages metadata for a POSIX-compliant shared file system
  - Directory hierarchy
  - File metadata (owner, timestamps, mode, etc.)
  - Stores metadata in RADOS
  - Does not access file content
  - Only required for shared file system
- The Ceph Metadata Server daemon (MDS) [1]
  - Provides the POSIX information needed by file systems that enables Ceph FS to interact with the Ceph Object Store
  - It remembers where data lives within a tree [2]
  - Clients accessing CephFS data first make a request to an MDS, which provides what they need to get files from the right OSDs [3]
- If you aren't running CephFS, MDS daemons do not need to be deployed.

© 2011-2014 Inktank Inc.

Note 1 : The MDS requires a 64bit OS because of the size of the INODES. This also means that ceph-fuse must be run also from a 64bit capable client

Note 2 : CephFS also keeps the recursive size of each directory that will appear at each level (. & .. Directory names)

There are 2 ways to mount a file system
1. The kernel based tool
2. The ceph-fuse tool (only alternative supported on all kernels that do not have the CephFS portion 2.6.32)

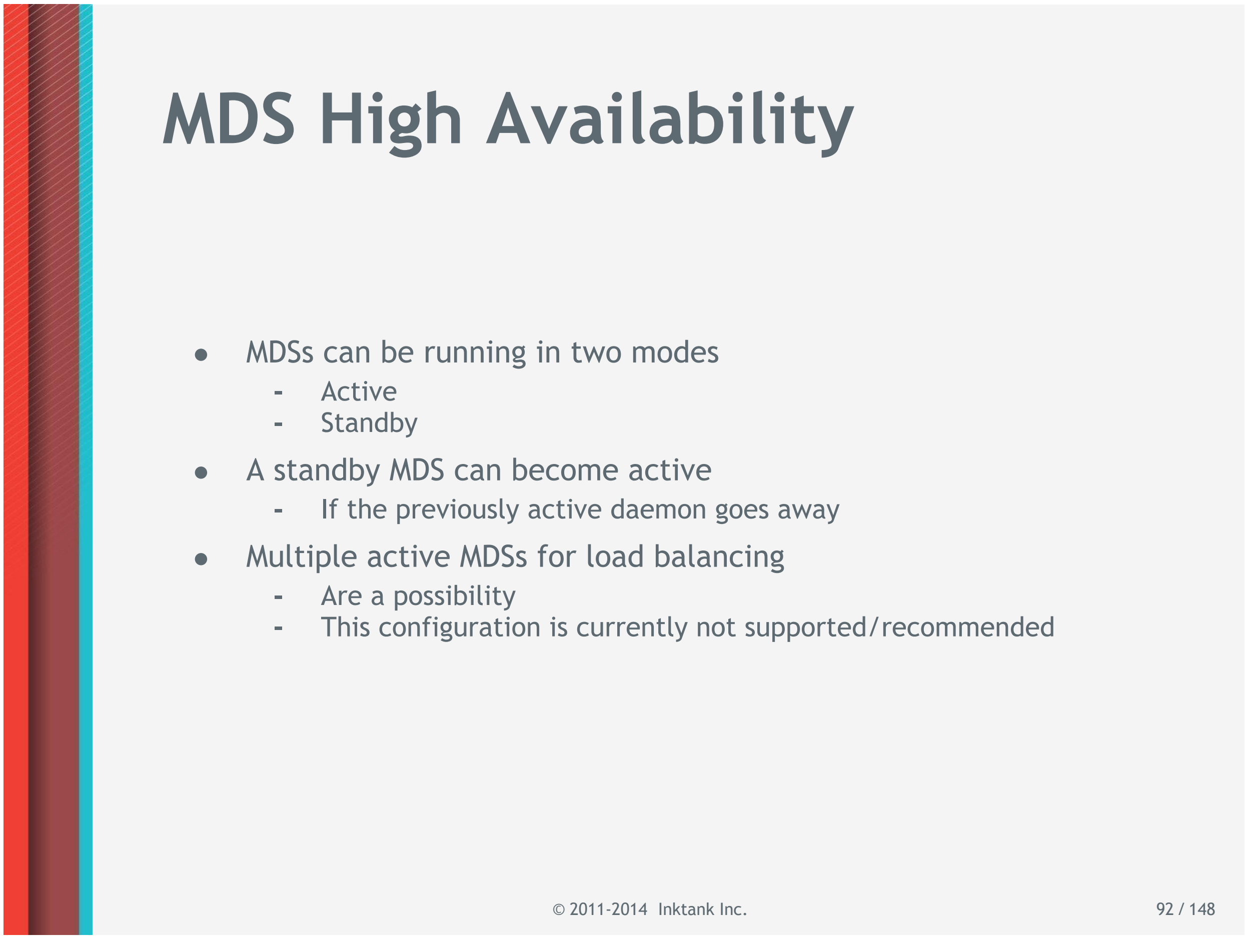
Ceph-fuse is most of the time slower than the CephFS kernel module

Note 3 : To mount with the kernel module, issue mount ¤t ceph <mon1,mon2, ...> making all MON running nodes are quoted for MON failure fault tolerance

To create a snapshot of a file system

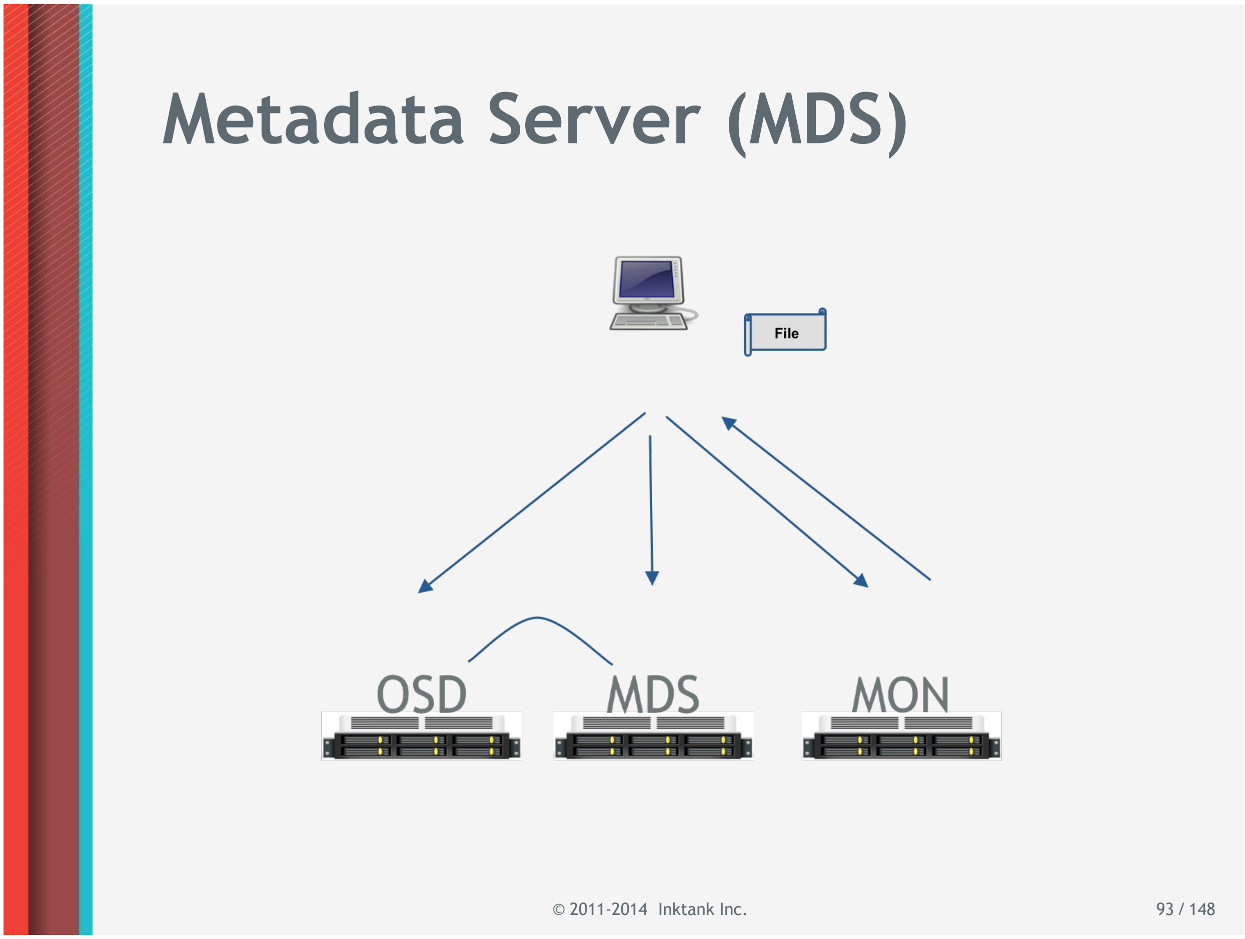
In the .snap directory of the file system, create a directory and that¤s it. From the file system root directory tree, issue mkdir ./.snap/snap_20131218_100000 command

To delete a snapshot of a file system, remove the corresponding snapshot directory name in the .snap directory and that¤s it. From the file system root directory tree, issue rm ./.snap/snap_20131218_100000 command

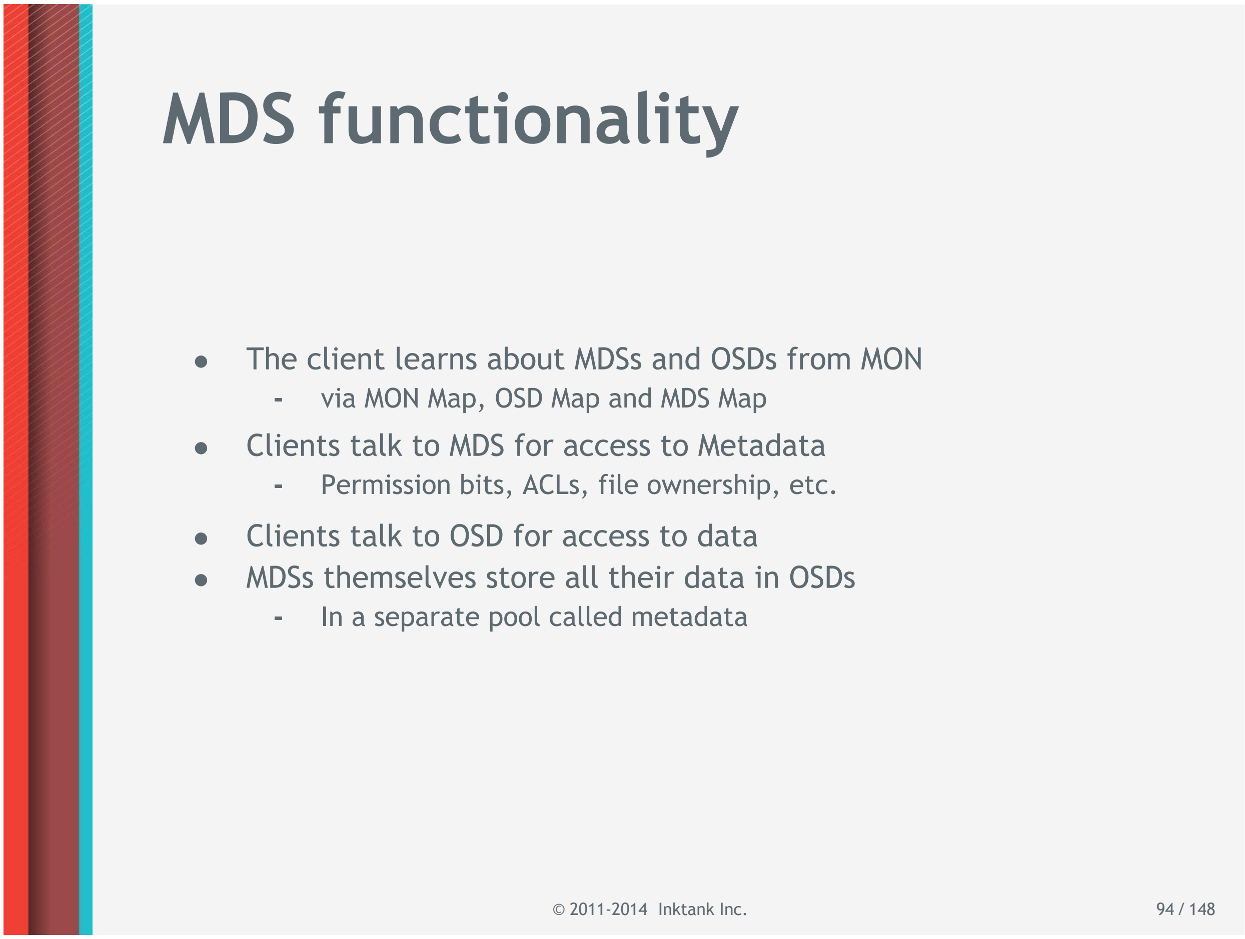


MDS High Availability

- MDSs can be running in two modes
  - Active
  - Standby
- A standby MDS can become active
  - If the previously active daemon goes away
- Multiple active MDSs for load balancing
  - Are a possibility
  - This configuration is currently not supported/recommended

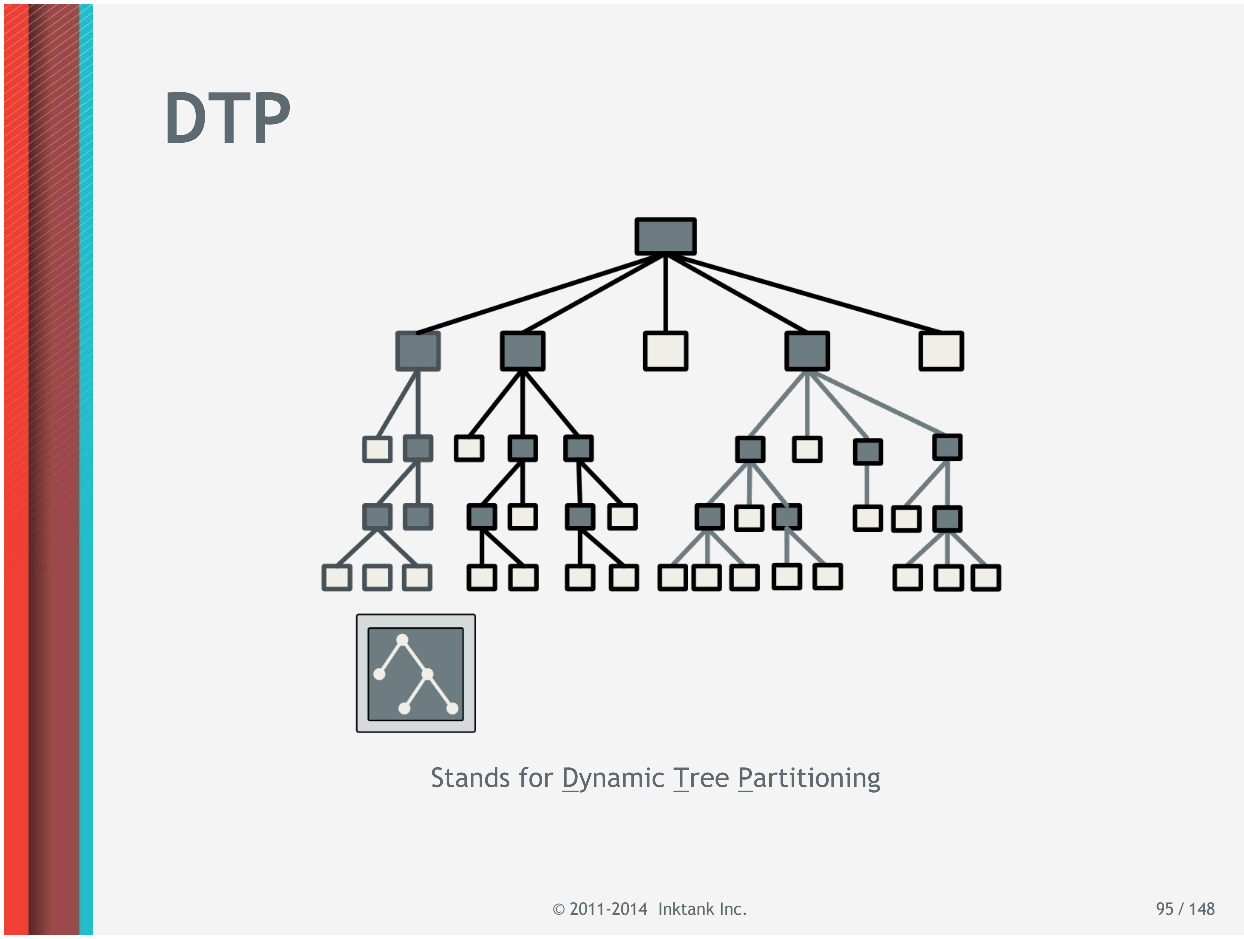


Metadata Server (MDS)



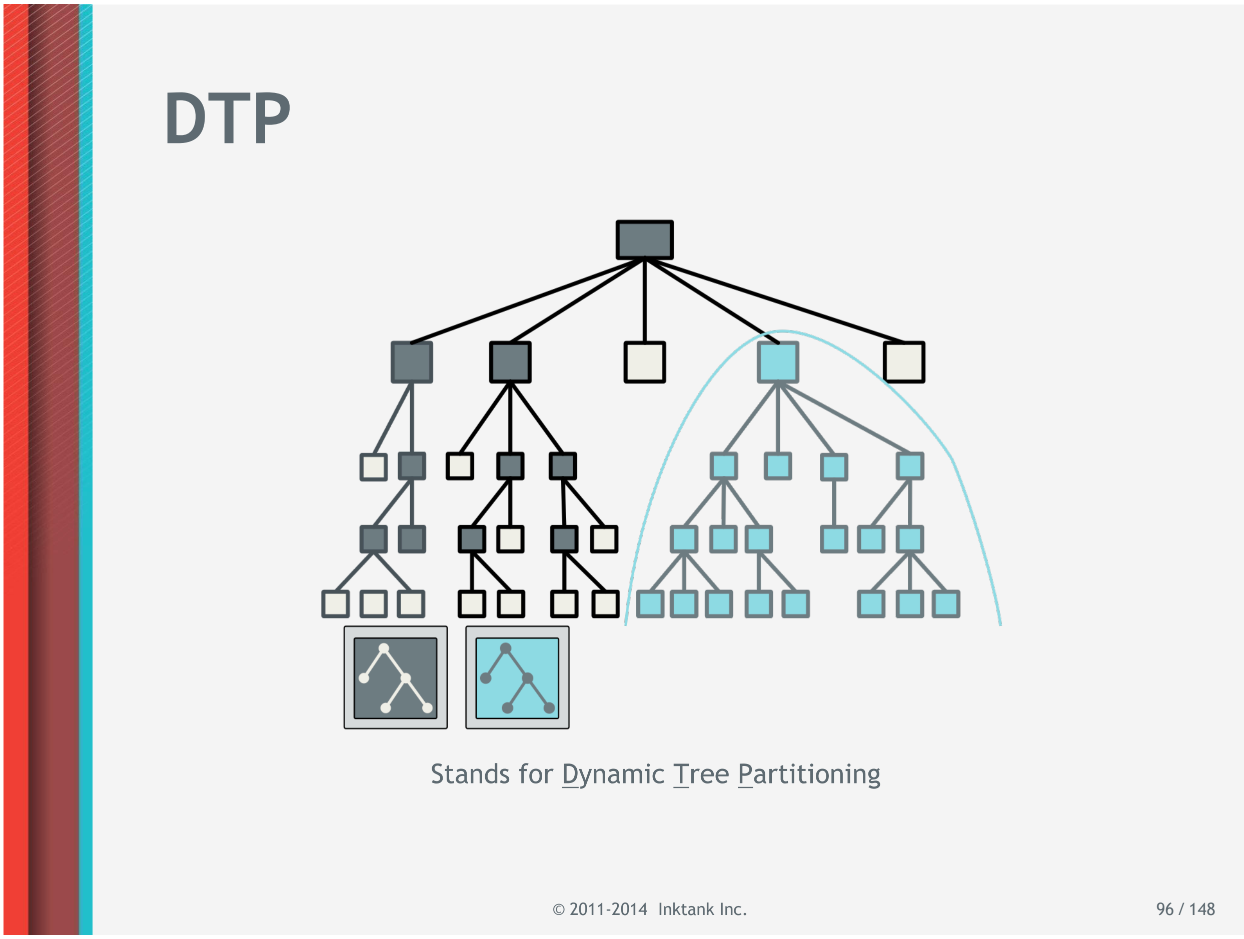
MDS functionality

- The client learns about MDSs and OSDs from MON
  - via MON Map, OSD Map and MDS Map
- Clients talk to MDS for access to Metadata
  - Permission bits, ACLs, file ownership, etc.
- Clients talk to OSD for access to data
- MDSs themselves store all their data in OSDs
  - In a separate pool called metadata



DTP

Stands for Dynamic Tree Partitioning

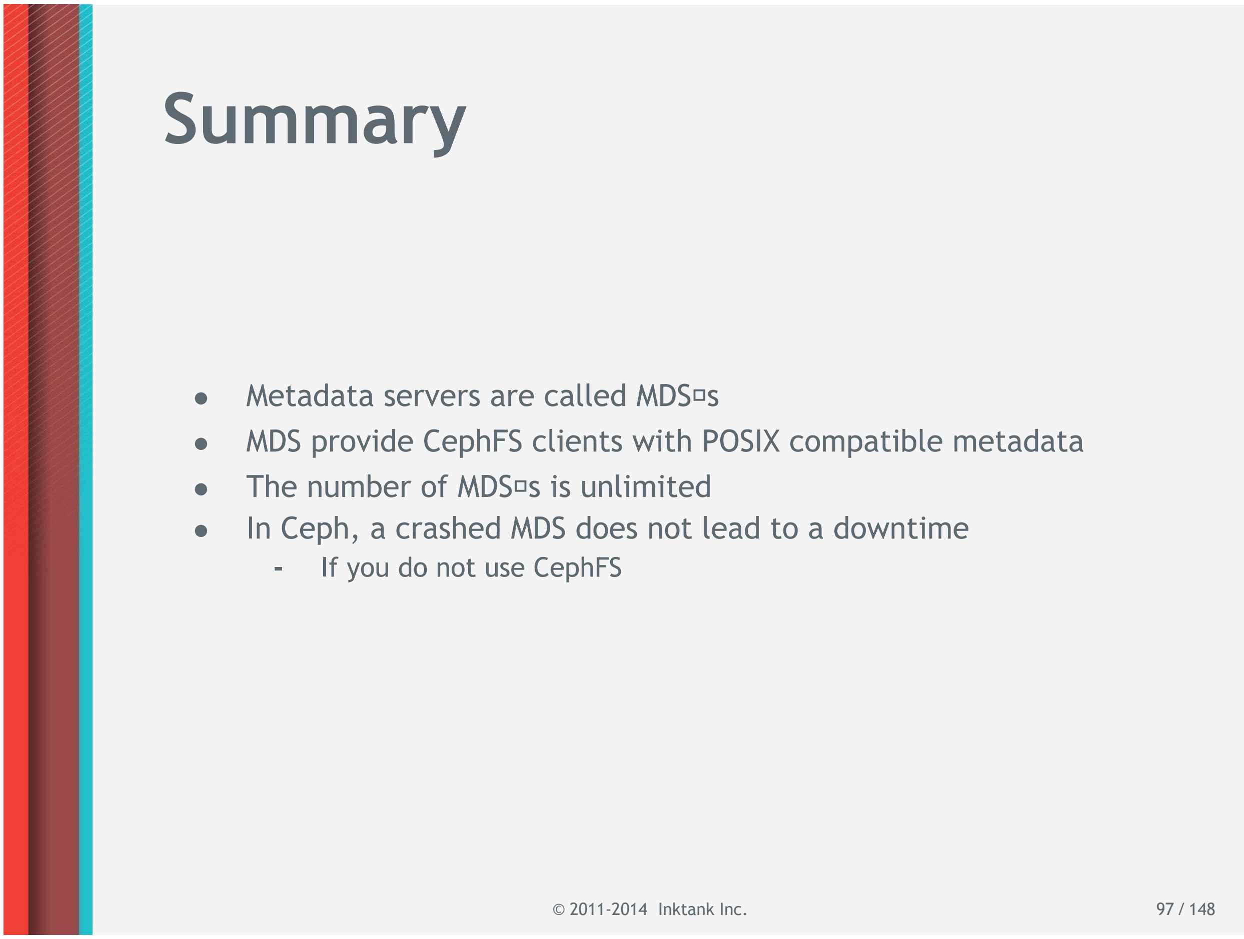


DTP

Stands for Dynamic Tree Partitioning

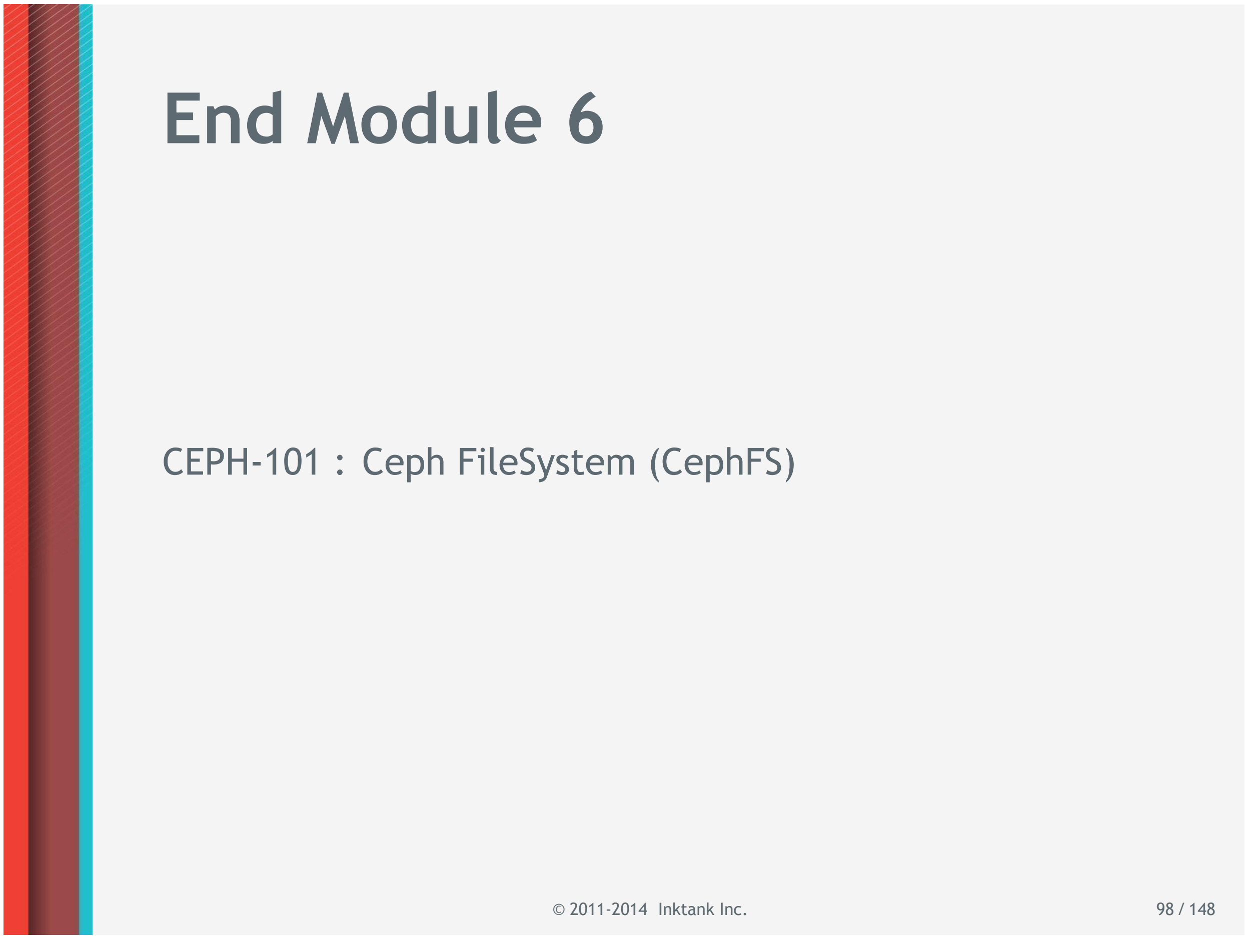
© 2011-2014 Inktank Inc.

96 / 148



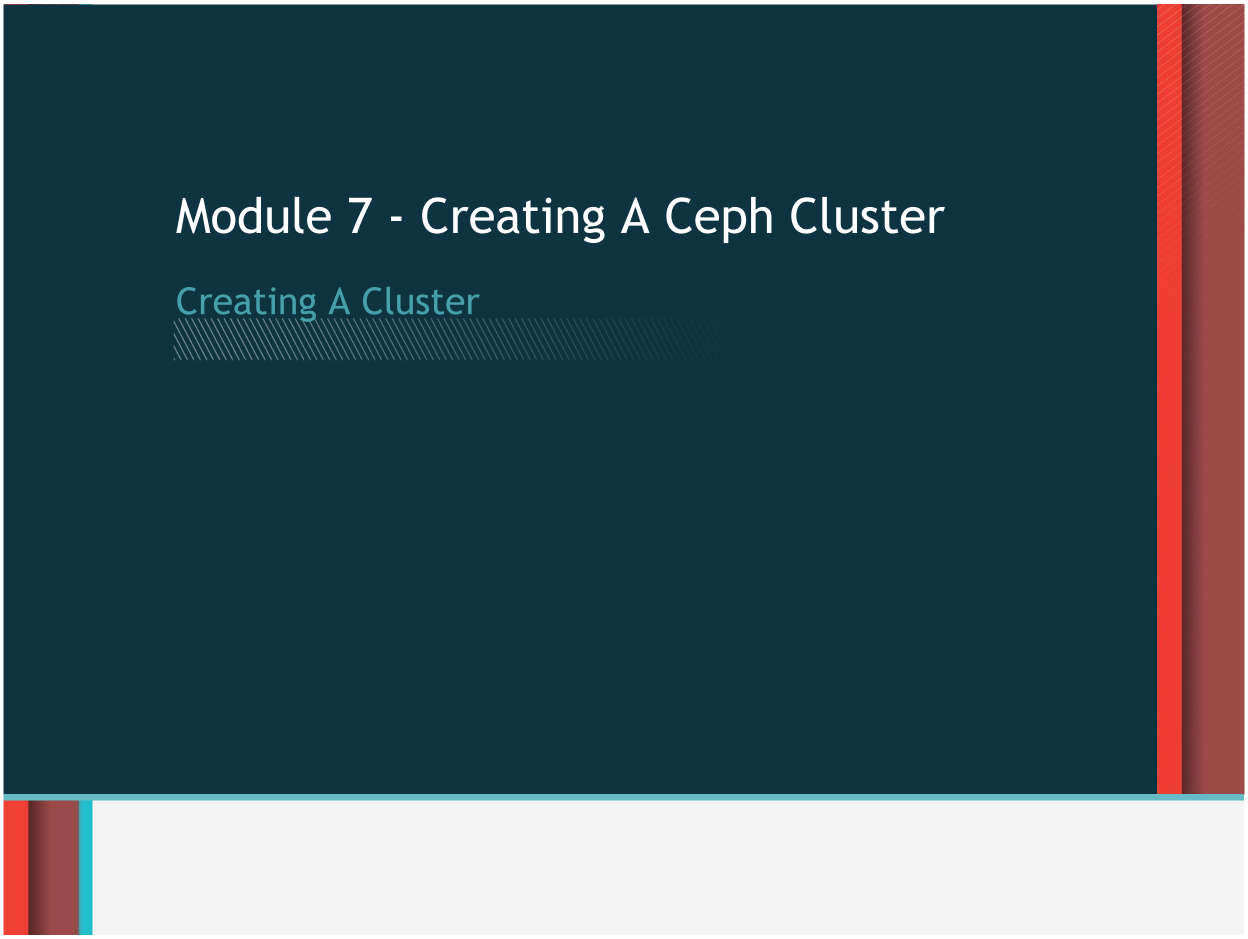
Summary

- Metadata servers are called MDS□s
- MDS provide CephFS clients with POSIX compatible metadata
- The number of MDS□s is unlimited
- In Ceph, a crashed MDS does not lead to a downtime
  - If you do not use CephFS



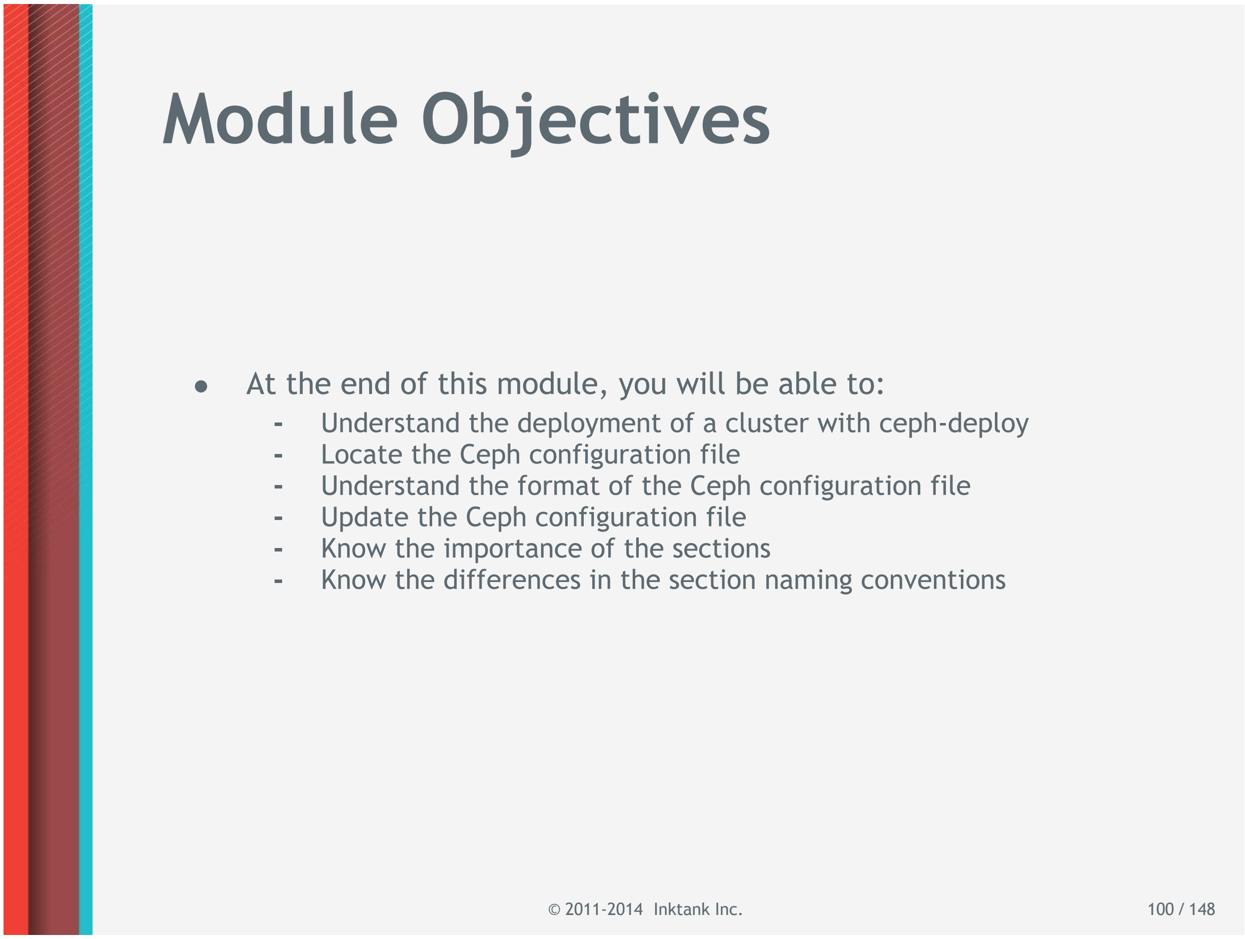
End Module 6

CEPH-101 : Ceph FileSystem (CephFS)



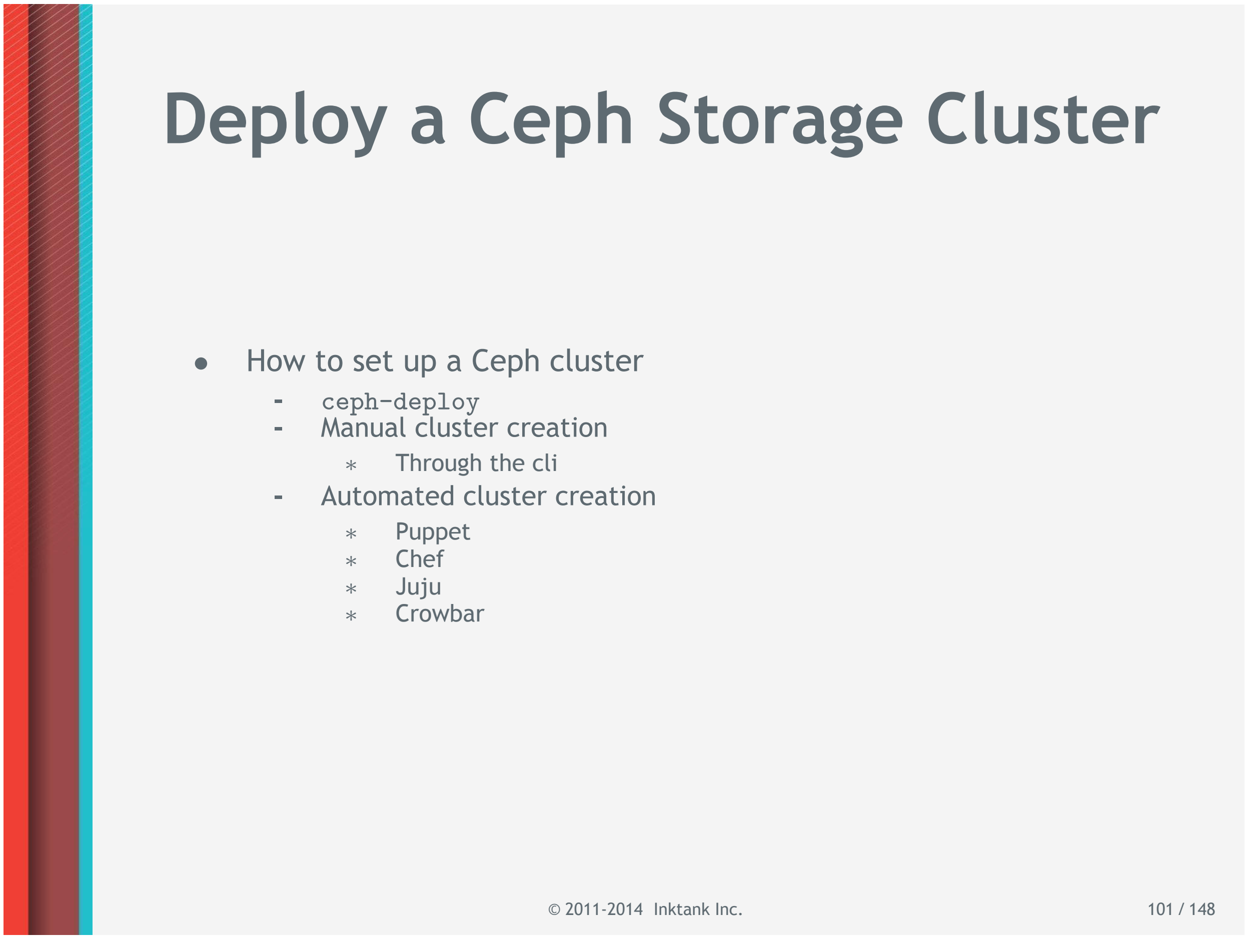
Module 7 - Creating A Ceph Cluster

Creating A Cluster



Module Objectives

- At the end of this module, you will be able to:
  - Understand the deployment of a cluster with ceph-deploy
  - Locate the Ceph configuration file
  - Understand the format of the Ceph configuration file
  - Update the Ceph configuration file
  - Know the importance of the sections
  - Know the differences in the section naming conventions

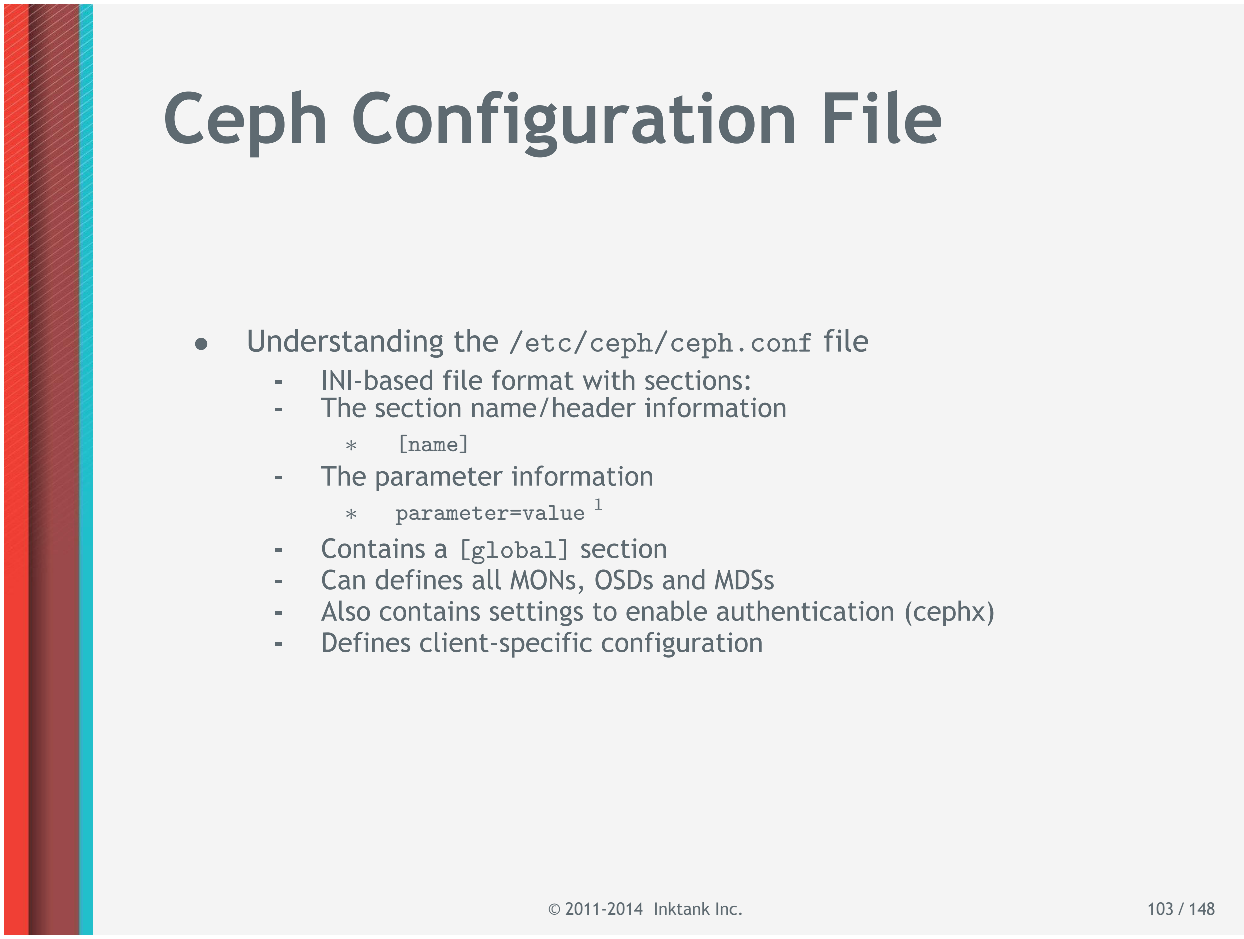


Deploy a Ceph Storage Cluster

- How to set up a Ceph cluster
  - `ceph-deploy`
  - Manual cluster creation
    - Through the cli
  - Automated cluster creation
    - Puppet
    - Chef
    - Juju
    - Crowbar

Creating a Ceph cluster

- Getting started
  - Ceph supports encrypted authentication (cephx)
  - Starting with Ceph 0.48, the default location for the per-daemon keyrings is `$datadir/keyring`,
    - datadir is defined with osd data, mds data, etc.
  - We recommend to always use the default paths
    - Udev hooks, ceph-disk and Upstart/Sysvinit scripts use those default path

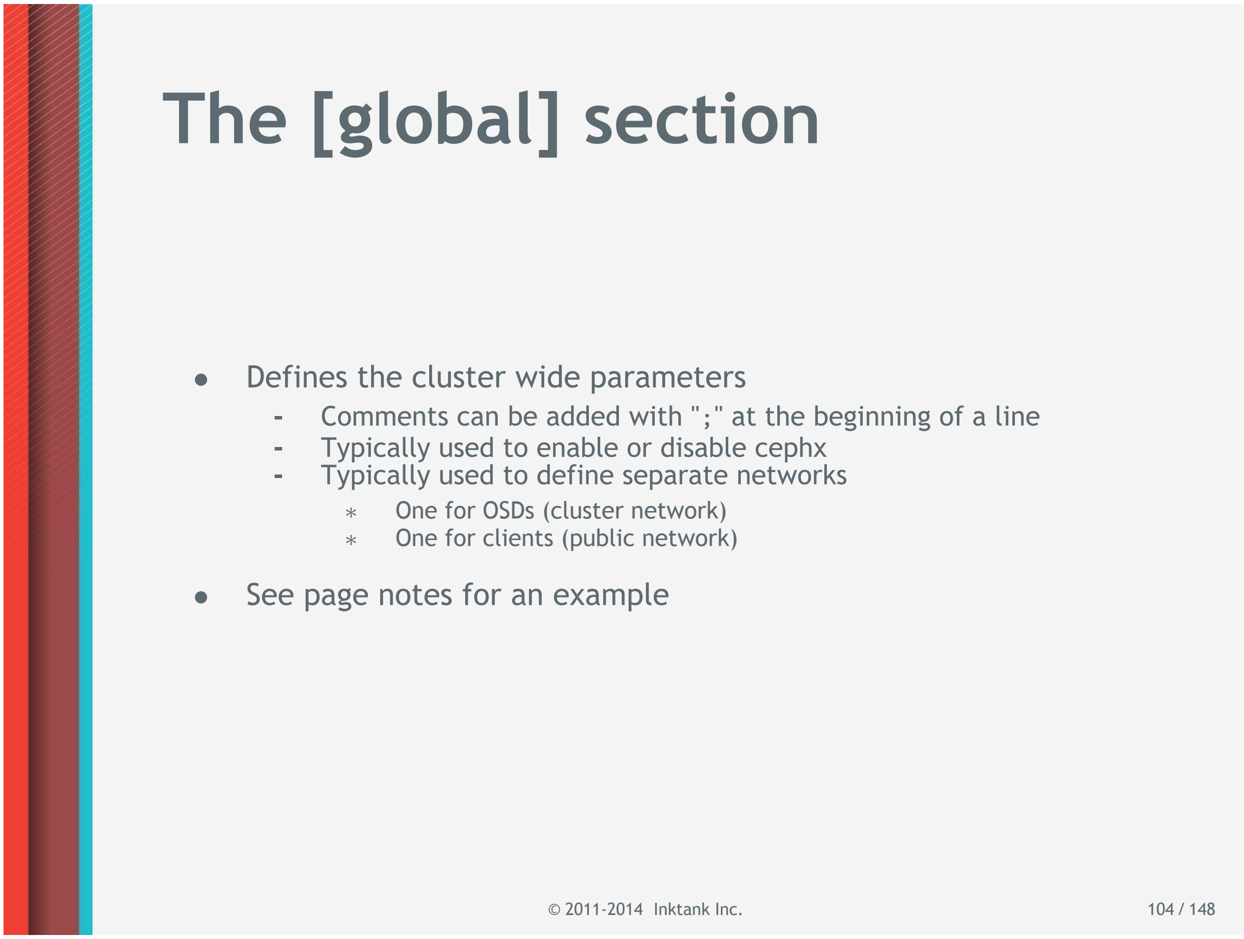


Ceph Configuration File

- Understanding the `/etc/ceph/ceph.conf` file
  - INI-based file format with sections:
  - The section name/header information
    - * `[name]`
  - The parameter information
    - * `parameter=value` [1]
  - Contains a `[global]` section
  - Can defines all MONs, OSDs and MDSs
  - Also contains settings to enable authentication (cephx)
  - Defines client-specific configuration

Note 1 : Parameter name can use space or _ as a separator

e.g. `osd journal size` or `osd_journal_size`



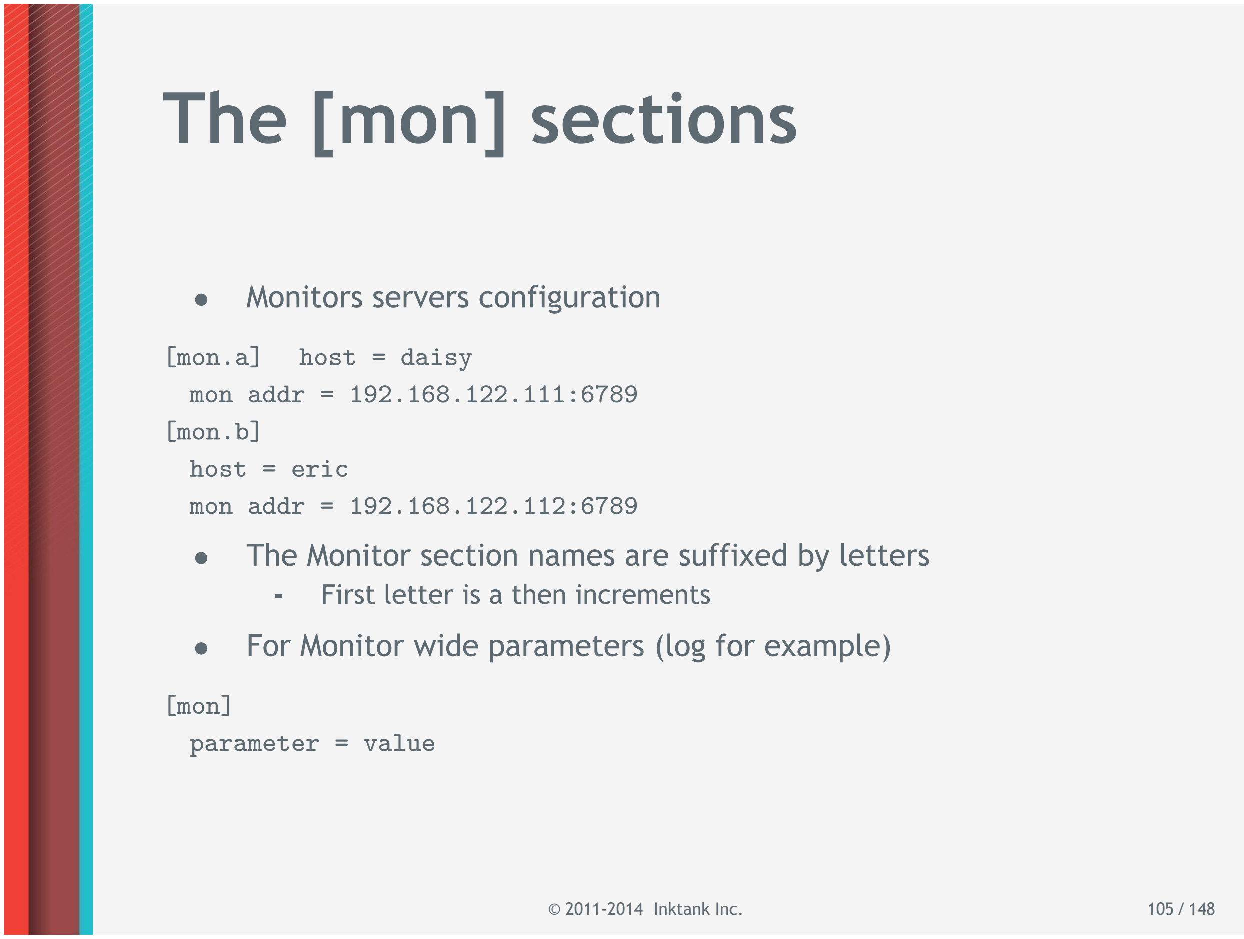
The [global] section

- Defines the cluster wide parameters
  - Comments can be added with ";" at the beginning of a line
  - Typically used to enable or disable cephx
  - Typically used to define separate networks
    - One for OSDs (cluster network)
    - One for clients (public network)
- See page notes for an example

Usually you use the global section to enable or disable general options such as cephx authenticaction

Cephx is the mechanism that will let you set permissions

```
[global]
  auth cluster required = cephx
  auth service required = cephx
  auth client required = cephx
  public network = {network}[, {network}]
  cluster network = {network}[, {network}]
  mon initial members = {hostname}[, {hostname}]
  mon host = {ip-address}[, {ip-address}]
  osd journal size = 1024
  filestore xattr use omap = true ; required for EXT4
```



The [mon] sections

- Monitors servers configuration

```
[mon.a]   host = daisy
  mon addr = 192.168.122.111:6789
[mon.b]
  host = eric
  mon addr = 192.168.122.112:6789
```

- The Monitor section names are suffixed by letters
  - First letter is a then increments
- For Monitor wide parameters (log for example)

```
[mon]
  parameter = value
```

Ceph-deploy build by default a default ceph.conf file

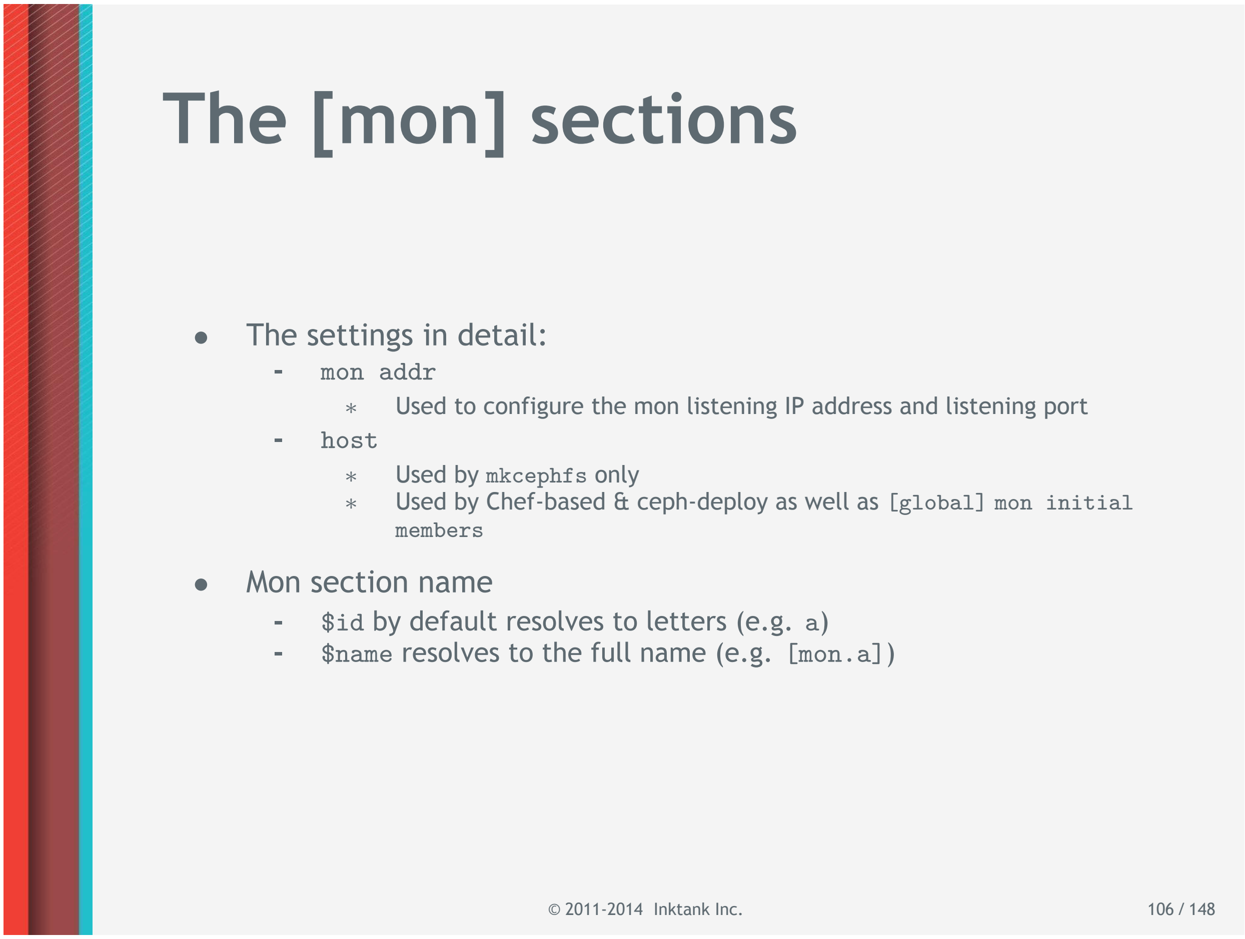
The global section can contain the list of the monitor members so that we can build the monitor map as soon as we have a quorum

mon_initial_members is a Best Practice parameter to use in the global section

Individual MON sections are often use for setting specific options such as debugging on a specific monitor

In production do not use ceph-deploy. Go through the manual deployment of monitors

urlhttp://ceph.com/docs/master/rados/operations/add-or-rm-mons/

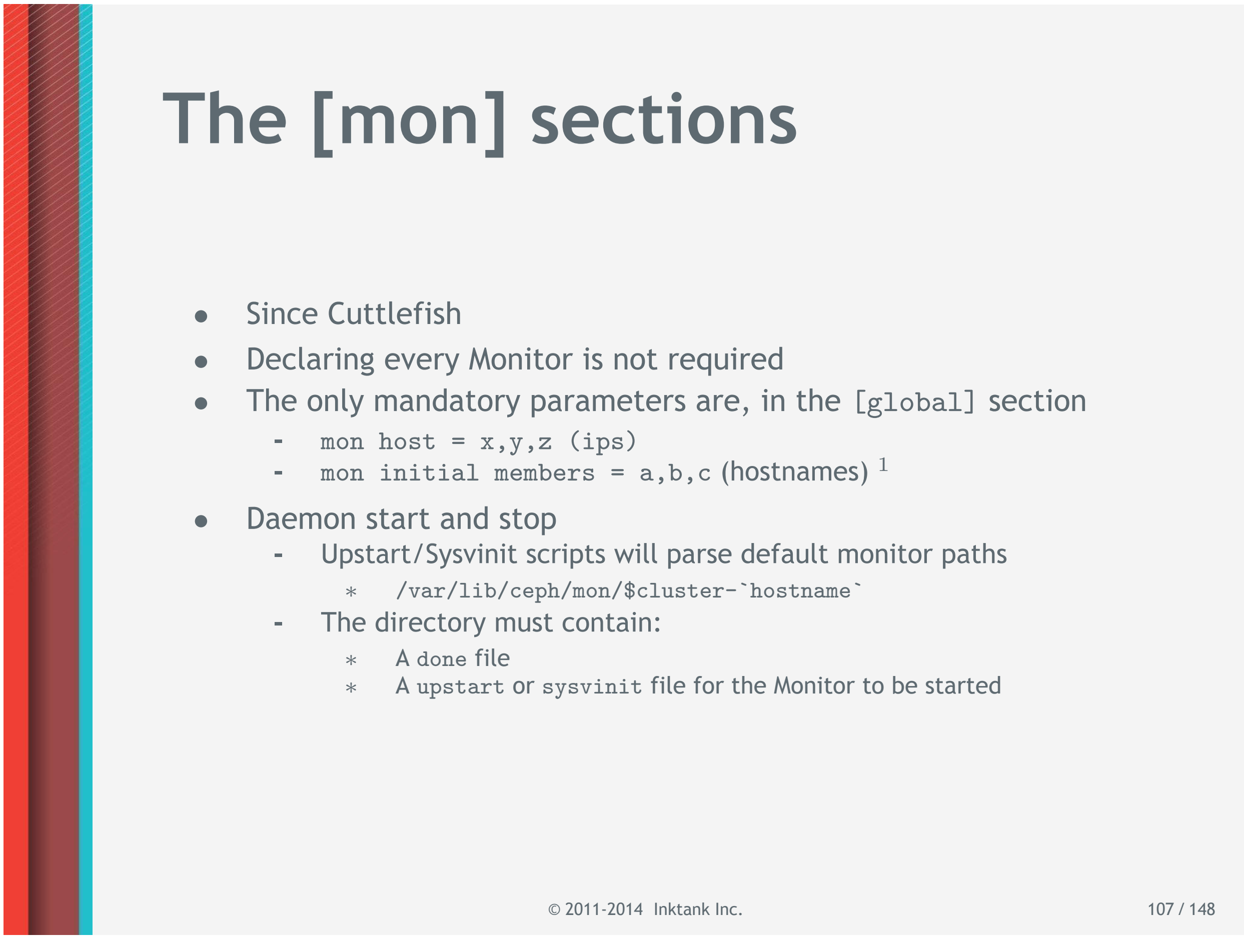


The [mon] sections

- The settings in detail:
  - `mon addr`
    - Used to configure the mon listening IP address and listening port
  - `host`
    - Used by `mkcephfs` only
    - Used by Chef-based & ceph-deploy as well as `[global] mon initial members`
- Mon section name
  - `$id` by default resolves to letters (e.g. `a`)
  - `$name` resolves to the full name (e.g. `[mon.a]`)

The host parameter is used by mkcephfs so you should not use it as this command is deprecated.

Keep the ceph.conf file as slim as possible



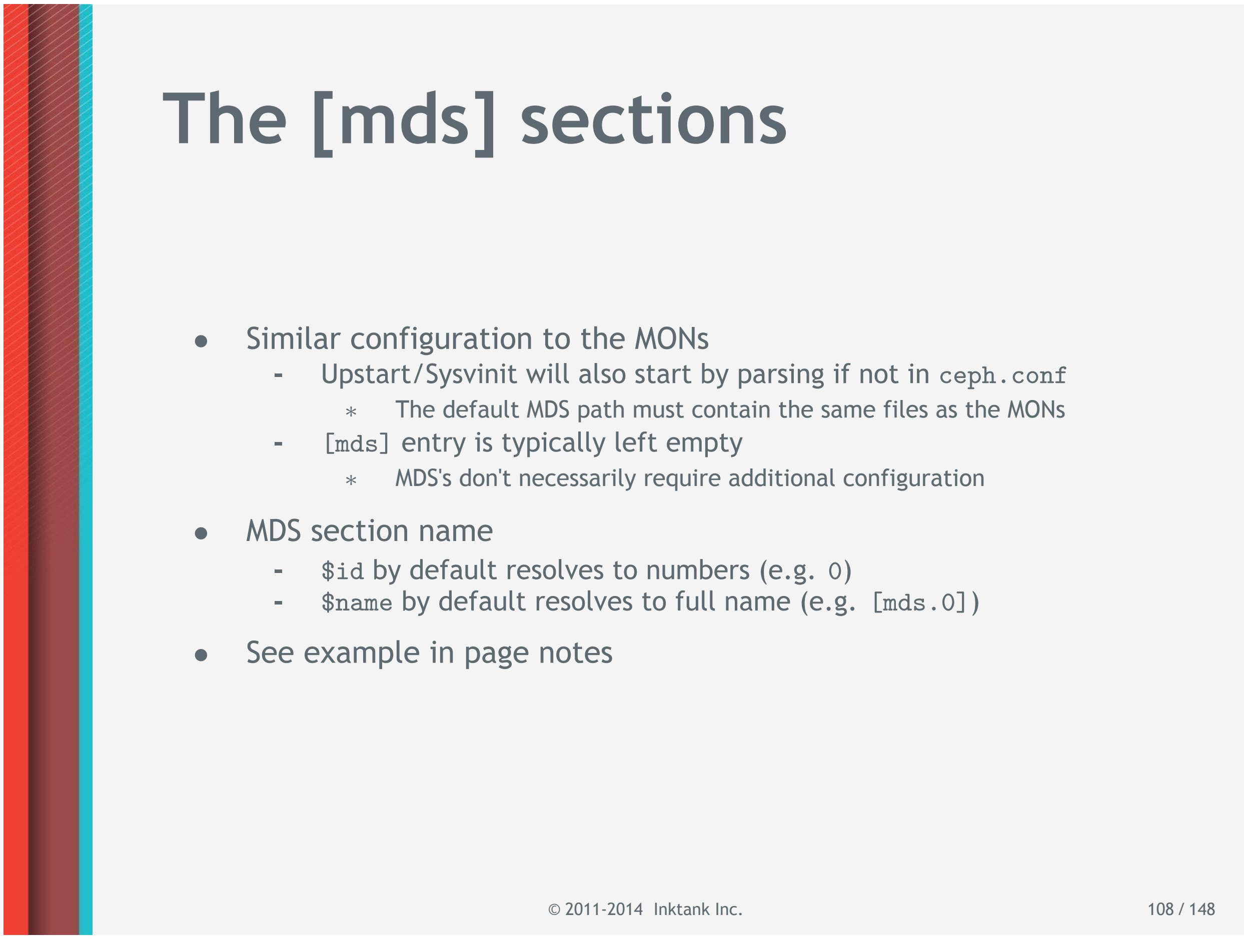
The [mon] sections

- Since Cuttlefish
- Declaring every Monitor is not required
- The only mandatory parameters are, in the `[global]` section
  - `mon host = x,y,z (ips)`
  - `mon initial members = a,b,c` (hostnames)[1]
- Daemon start and stop
  - Upstart/Sysvinit scripts will parse default monitor paths
    - `/var/lib/ceph/mon/$cluster-`hostname``
  - The directory must contain:
    - A `done` file
    - A `upstart` or `sysvinit` file for the Monitor to be started

Note 1 : The mon_initial_members parameter avoids a brain split during the first start making sure quorum is gained as soon as possible

The default install path is: /var/lib/ceph/mon/$cluster-`hostname`

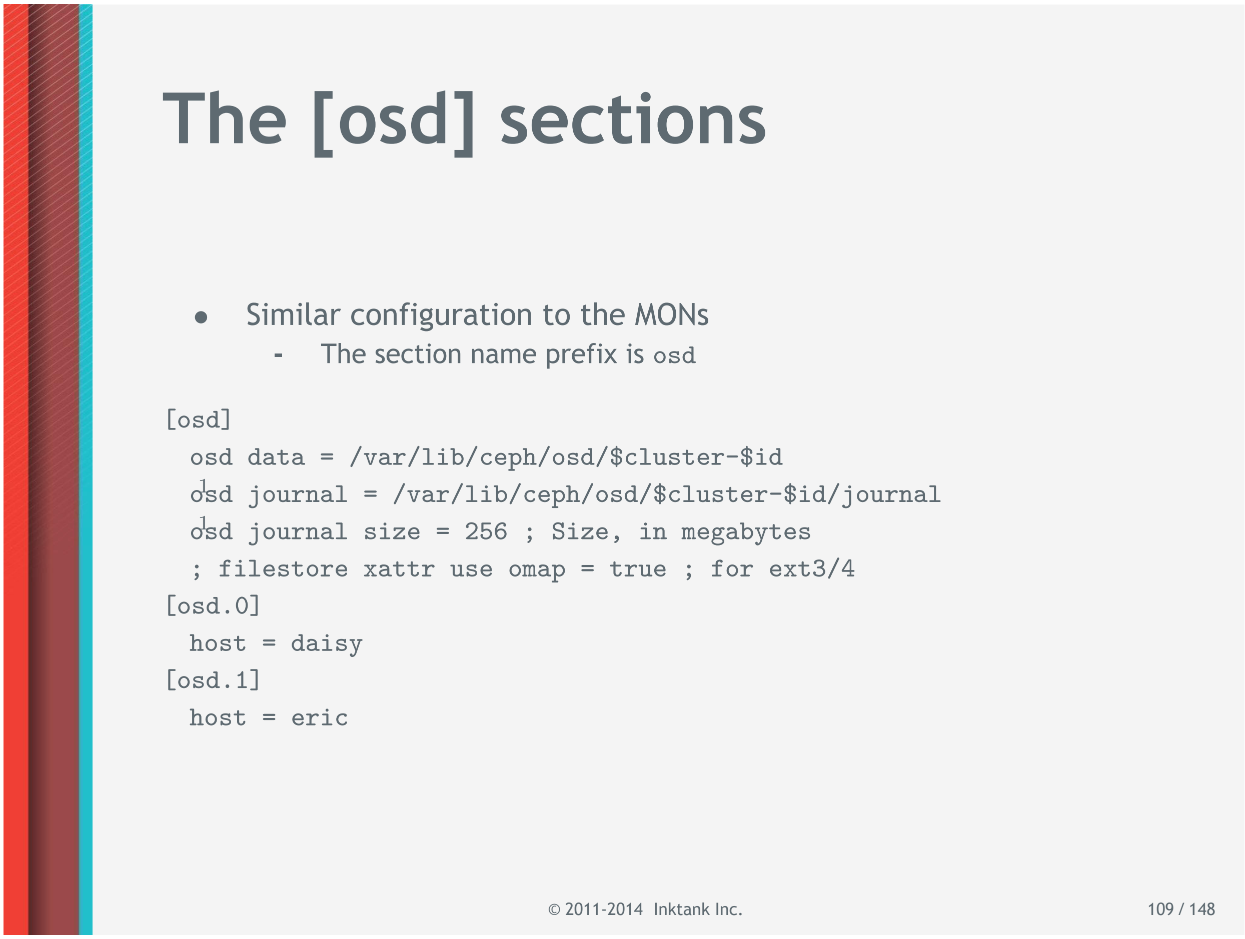
The best practice is to use the default path



The [mds] sections

- Similar configuration to the MONs
  - Upstart/Sysvinit will also start by parsing if not in `ceph.conf`
    - The default MDS path must contain the same files as the MONs
  - `[mds]` entry is typically left empty
    - MDS's don't necessarily require additional configuration
- MDS section name
  - `$id` by default resolves to numbers (e.g. `0`)
  - `$name` by default resolves to full name (e.g. `[mds.0]`)
- See example in page notes

```
[mds.0]
  host = daisy
[mds.1]
  host = eric
```



The [osd] sections

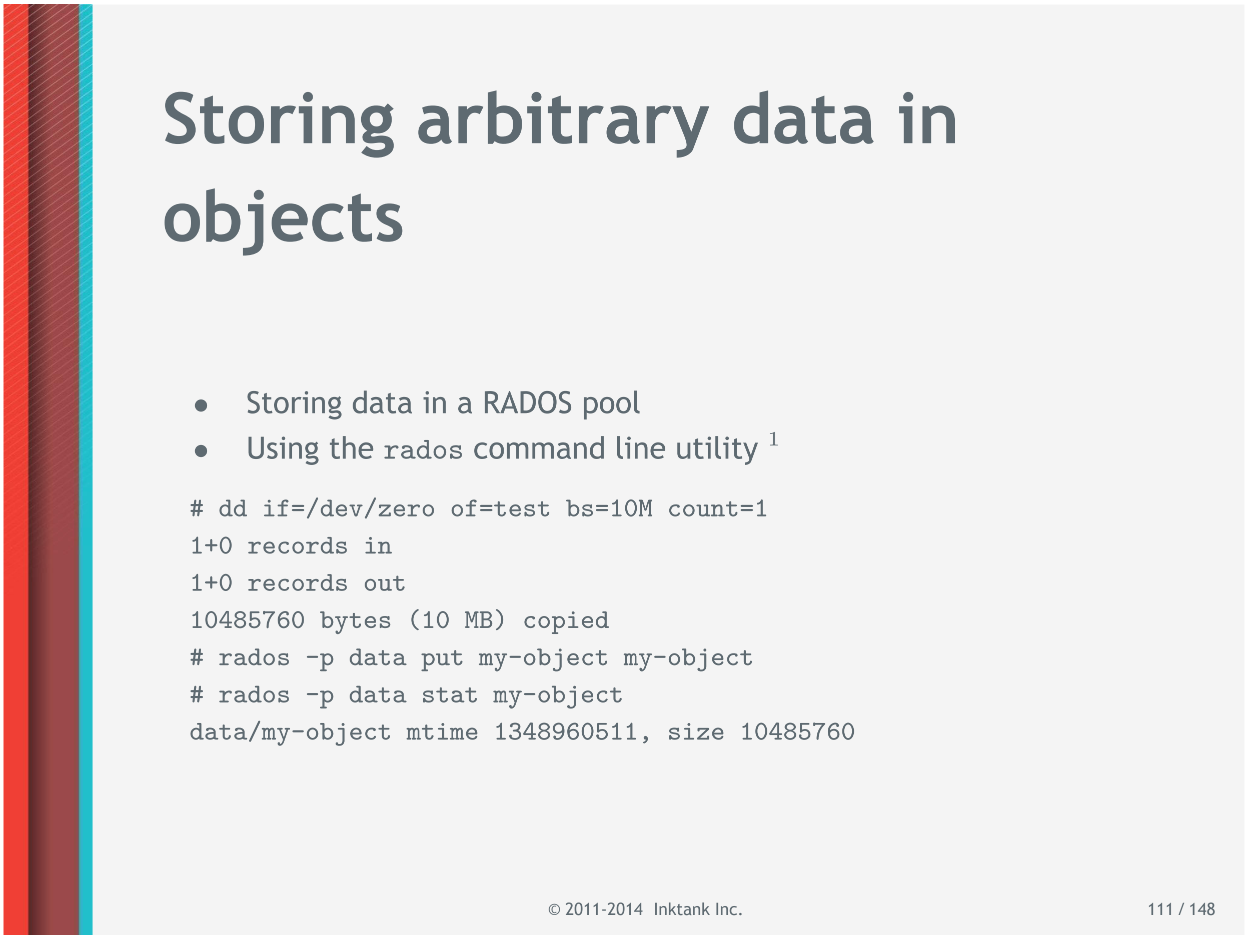
- Similar configuration to the MONs
  - The section name prefix is `osd`

```
[osd]
  osd data = /var/lib/ceph/osd/$cluster-$id
  osd[1] journal = /var/lib/ceph/osd/$cluster-$id/journal
  osd[1] journal size = 256 ; Size, in megabytes
  ; filestore xattr use omap = true ; for ext3/4
[osd.0]
  host = daisy
[osd.1]
  host = eric
```

Note 1 : If the journal is to be changed:
1. Stop the OSD
2. Modify the file
3. ceph osd ¤i ¤mkjournal to start a new journal
4. Restart the OSD

The [osd] sections

- Configuring OSDs
  - the data and journal settings here resemble the standard
  - and are thus redundant
  - for educational purposes in this example only
  - Default value for `$cluster` is `ceph`
  - When using `EXT3` and `EXT4` as a file store
    - `xattr use omap = true` is a requirement
- OSD section name
  - `$id` by default resolves to numbers (e.g. `0`)
  - `$name` by default resolves to full name (e.g. `[osd.0@]`)
- Journal parameters
  - Can point to a fast block device rather than a file (SSDs)
  - ceph-deploy puts journal and data on the same device by default

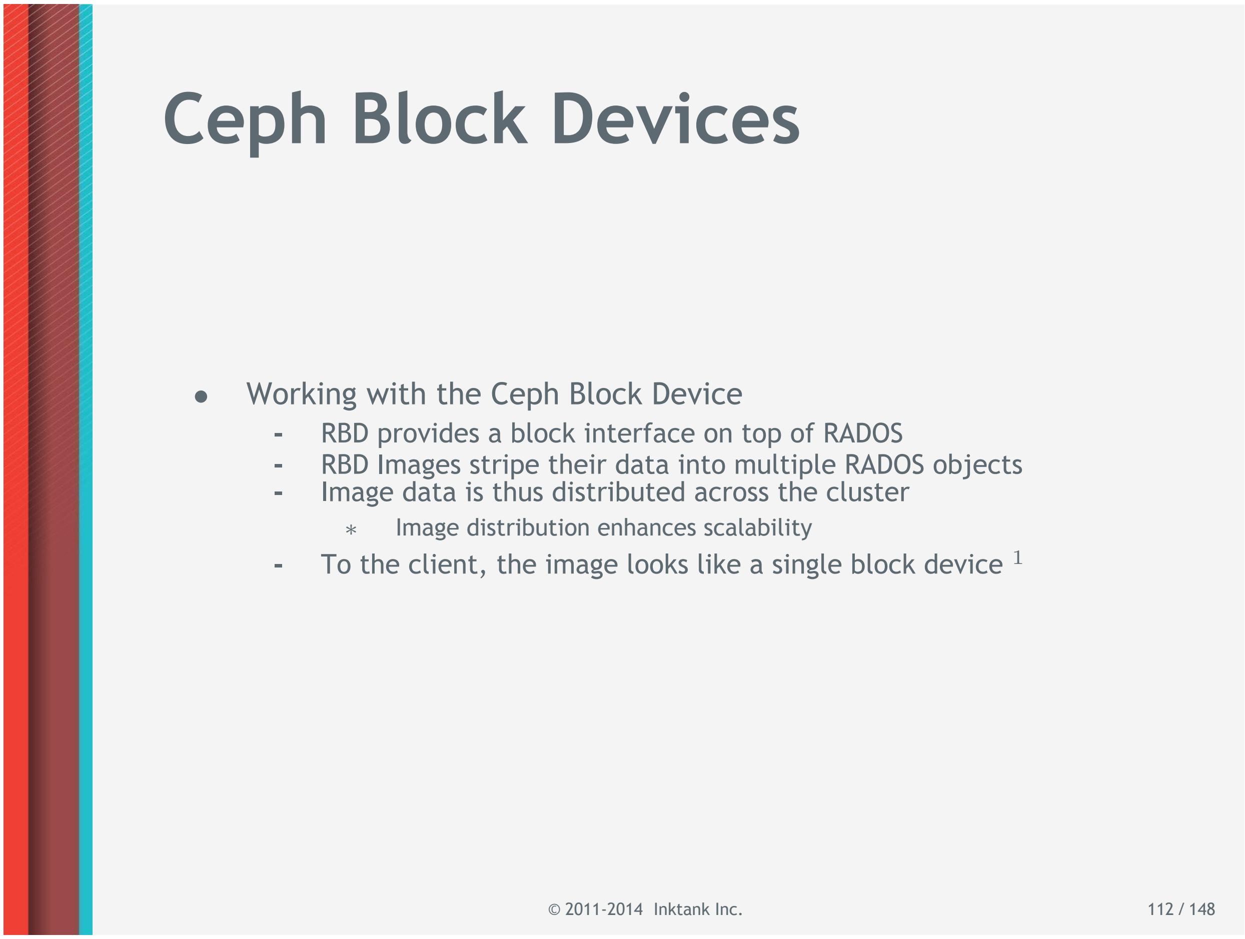


Storing arbitrary data in objects

- Storing data in a RADOS pool
- Using the `rados` command line utility [1]

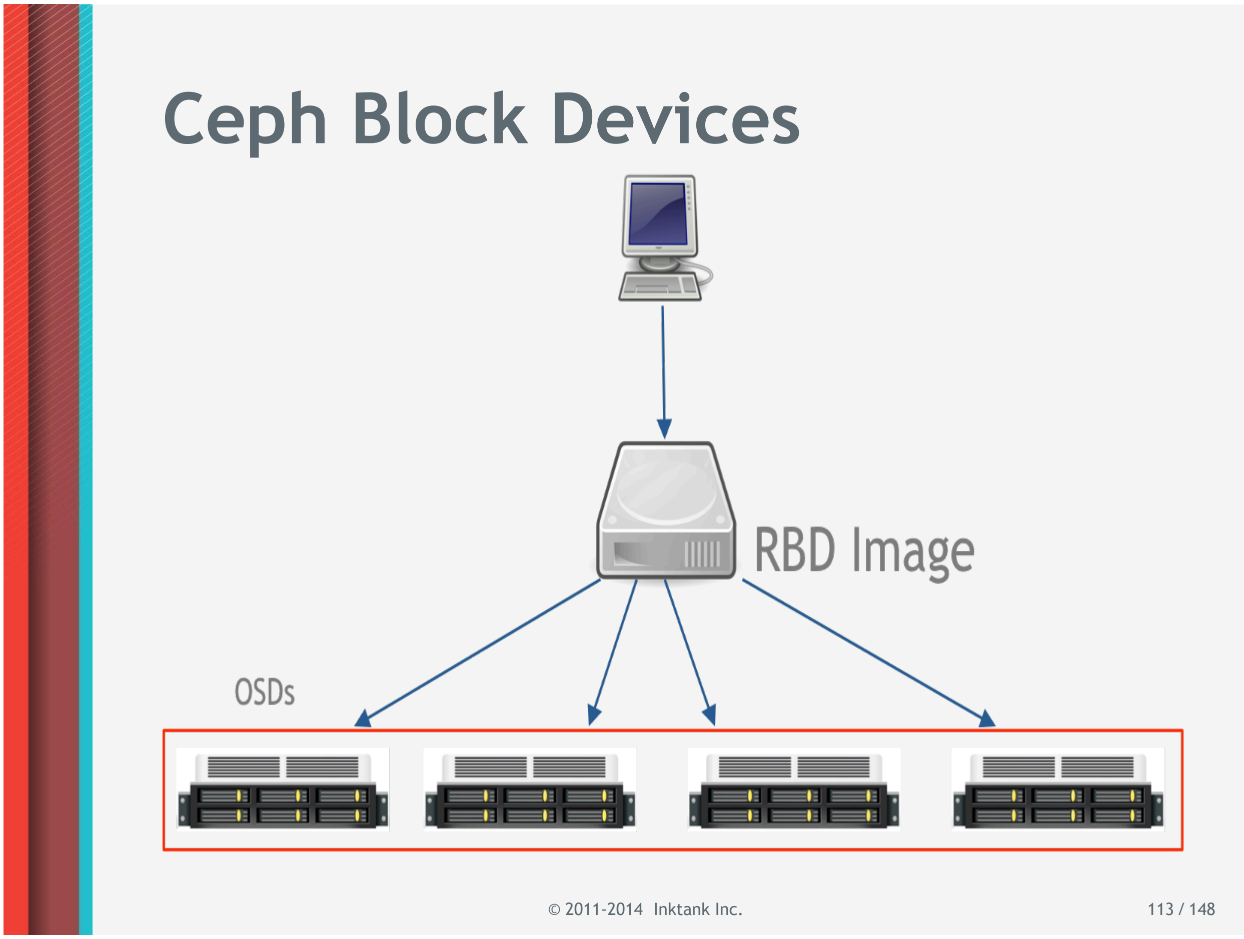
```
# dd if=/dev/zero of=test bs=10M count=1
1+0 records in
1+0 records out
10485760 bytes (10 MB) copied
# rados -p data put my-object my-object
# rados -p data stat my-object
data/my-object mtime 1348960511, size 10485760
```

Note 1 : RADOS does no implement striping.



Ceph Block Devices

- Working with the Ceph Block Device
  - RBD provides a block interface on top of RADOS
  - RBD Images stripe their data into multiple RADOS objects
  - Image data is thus distributed across the cluster
    - Image distribution enhances scalability
  - To the client, the image looks like a single block device [1]



Ceph Block Devices

© 2011-2014 Inktank Inc.

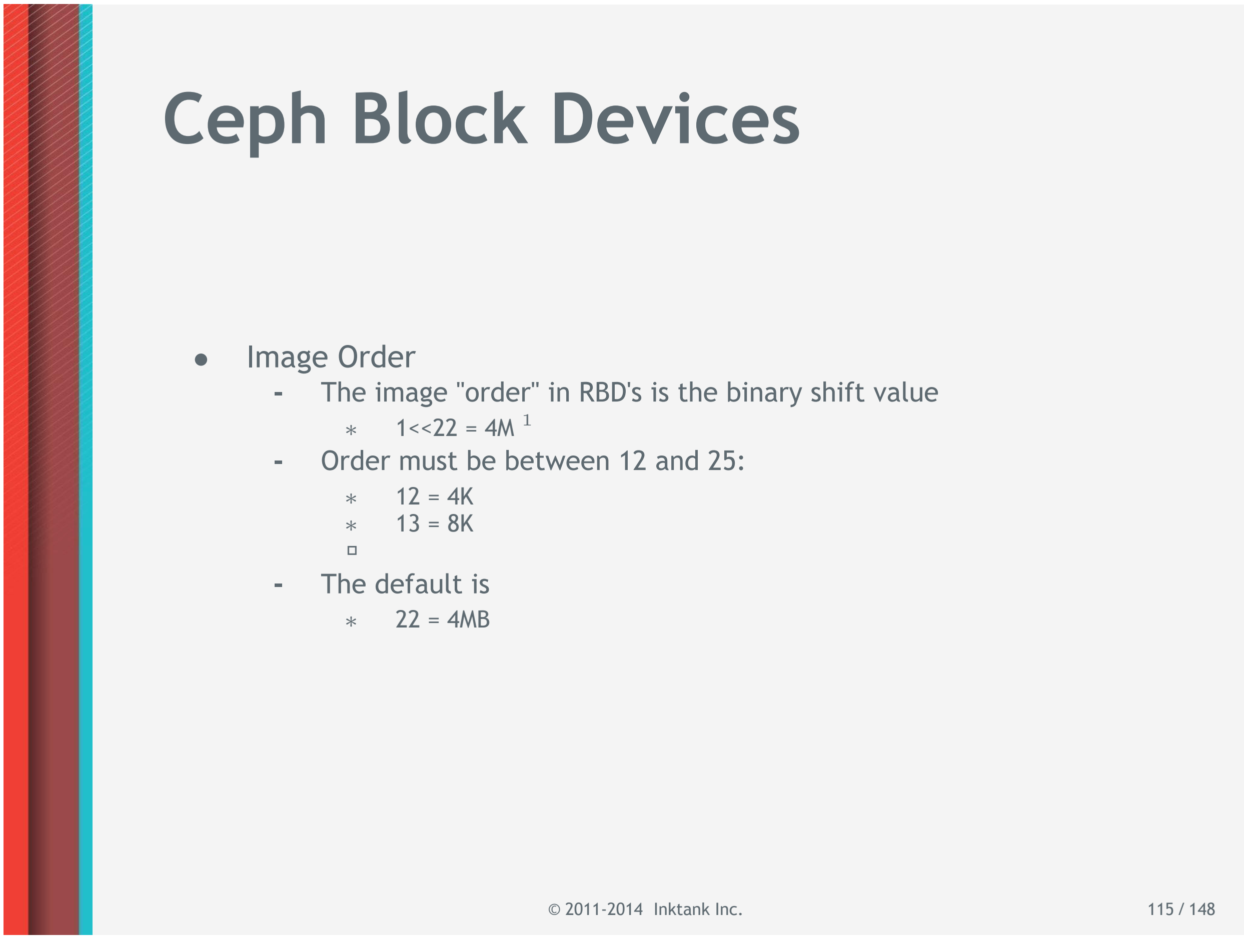
113 / 148

As you can see in this slide, data coming from a client to an RBD image will be split among the various OSD processes and underlying drives. As explained on the previous slide.

Ceph Block Devices

- Creating an RBD image

```
# rbd create --size 1024 foo
# rbd ls
foo
# rbd info foo
rbd image foo:
size 1024 MB in 256 objects
order 22 (4096 KB objects)
block_name_prefix: rb.0.0
parent:  (pool -1)
```



Ceph Block Devices

- Image Order
  - The image "order" in RBD's is the binary shift value
    - 1<<22 = 4M[1]
  - Order must be between 12 and 25:
    - 12 = 4K
    - 13 = 8K
    - □
  - The default is
    - 22 = 4MB

Note 1 : The `<<` C operator is a left bit shifting operator. `<<` shifts the left operand bits by the right operand value
A binary value for example

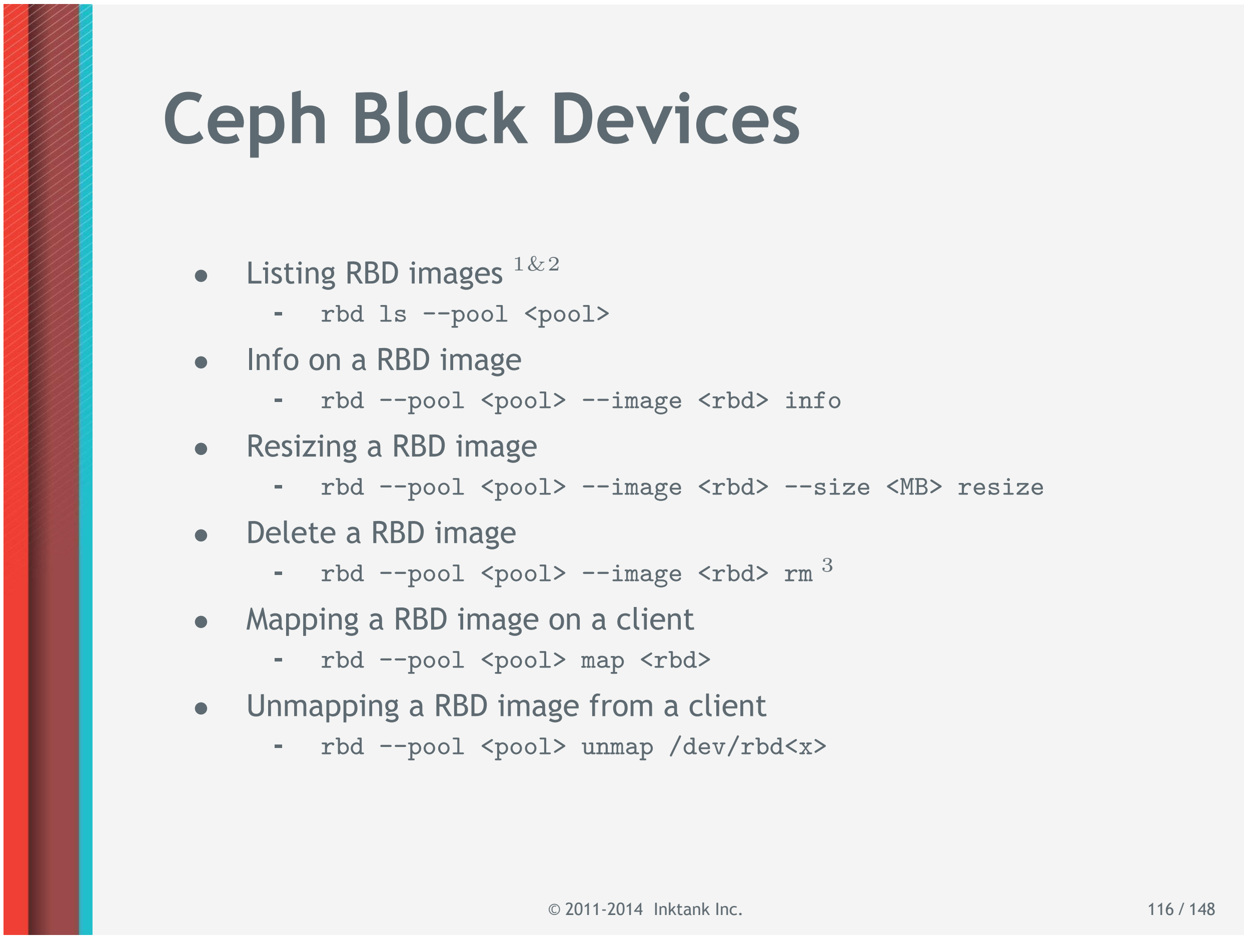
```
1 = 0001
```

If we do `1 << 2` the resulting value is

```
4 = 0100
```

The opposite operator is `>>`, the right bit shifting operator

The advantage of these operators is that they are executed in a single CPU cycle



Ceph Block Devices

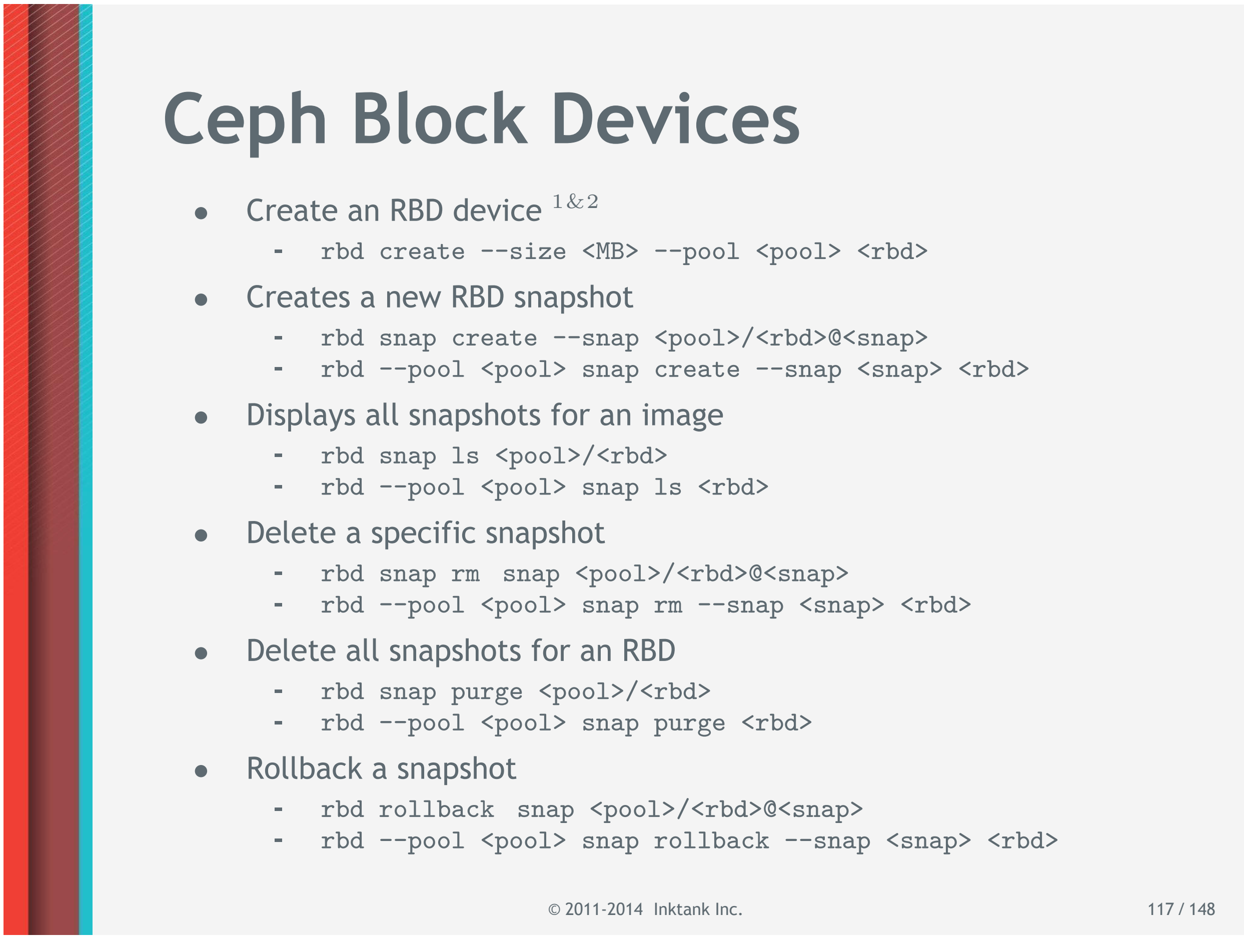
- Listing RBD images $^{1\&2}$
  - `rbd ls --pool <pool>`
- Info on a RBD image
  - `rbd --pool <pool> --image <rbd> info`
- Resizing a RBD image
  - `rbd --pool <pool> --image <rbd> --size <MB> resize`
- Delete a RBD image
  - `rbd --pool <pool> --image <rbd> rm` ^{3}
- Mapping a RBD image on a client
  - `rbd --pool <pool> map <rbd>`
- Unmapping a RBD image from a client
  - `rbd --pool <pool> unmap /dev/rbd<x>`

© 2011-2014 Inktank Inc. 116 / 148

Note 1 : The pool defaults to rbd if omitted

Note 2 : --pool can be replaced with ¤p

Note 3 : Deleting the RBD image will fail if snapshots still exist for this particular image. Hence in this case, the snap purge command will have to be issued first.



Ceph Block Devices

- Create an RBD device [1&2]

```
- rbd create --size <MB> --pool <pool> <rbd>
```

- Creates a new RBD snapshot

```
- rbd snap create --snap <pool>/<rbd>@<snap>
- rbd --pool <pool> snap create --snap <snap> <rbd>
```

- Displays all snapshots for an image

```
- rbd snap ls <pool>/<rbd>
- rbd --pool <pool> snap ls <rbd>
```

- Delete a specific snapshot

```
- rbd snap rm  snap <pool>/<rbd>@<snap>
- rbd --pool <pool> snap rm --snap <snap> <rbd>
```

- Delete all snapshots for an RBD

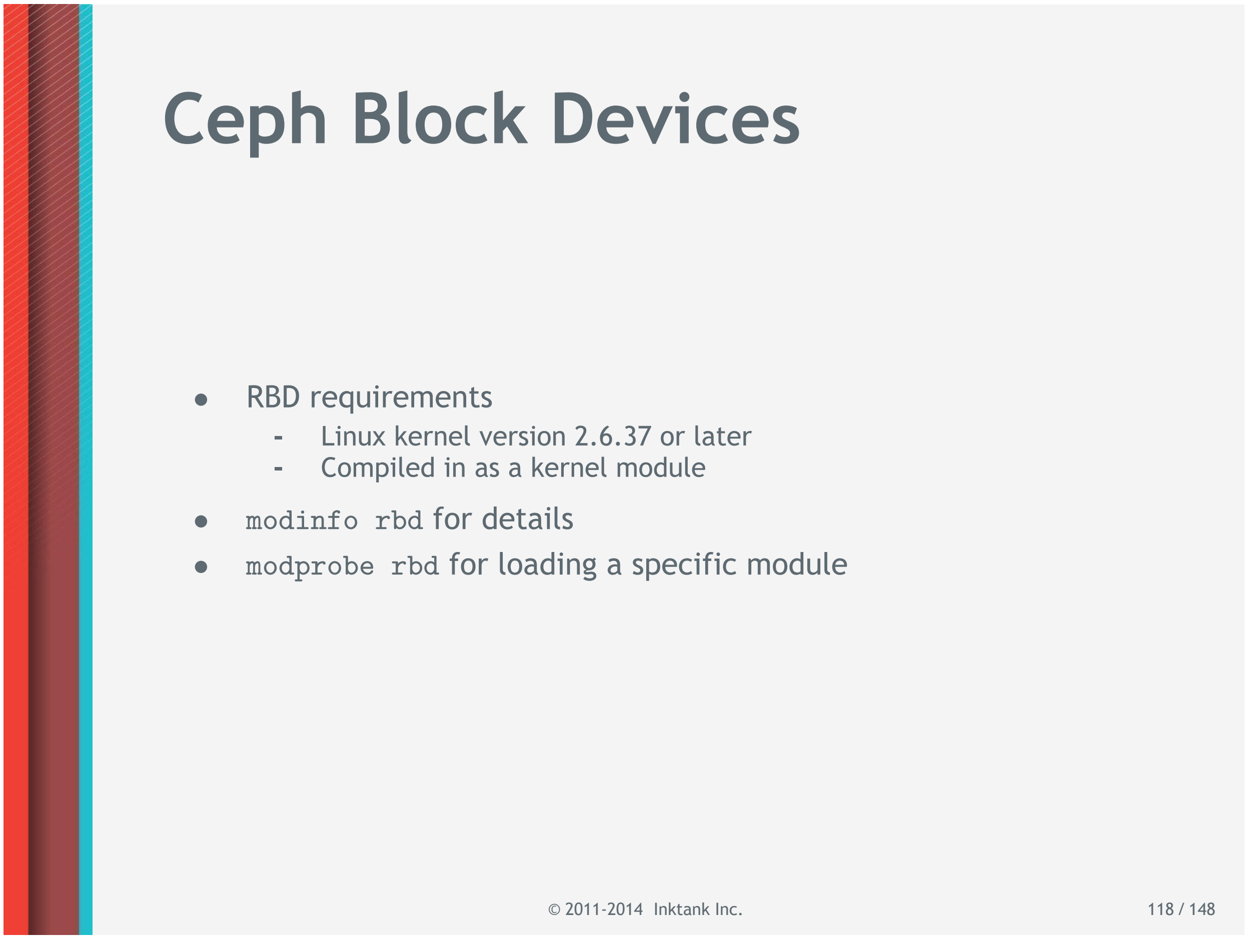
```
- rbd snap purge <pool>/<rbd>
- rbd --pool <pool> snap purge <rbd>
```

- Rollback a snapshot

```
- rbd rollback  snap <pool>/<rbd>@<snap>
- rbd --pool <pool> snap rollback --snap <snap> <rbd>
```

Note 1 : The pool defaults to rbd if omitted

Note 2 : --pool option can be replaced with the ▫p option



Ceph Block Devices

- RBD requirements
  - Linux kernel version 2.6.37 or later
  - Compiled in as a kernel module
- `modinfo rbd` for details
- `modprobe rbd` for loading a specific module

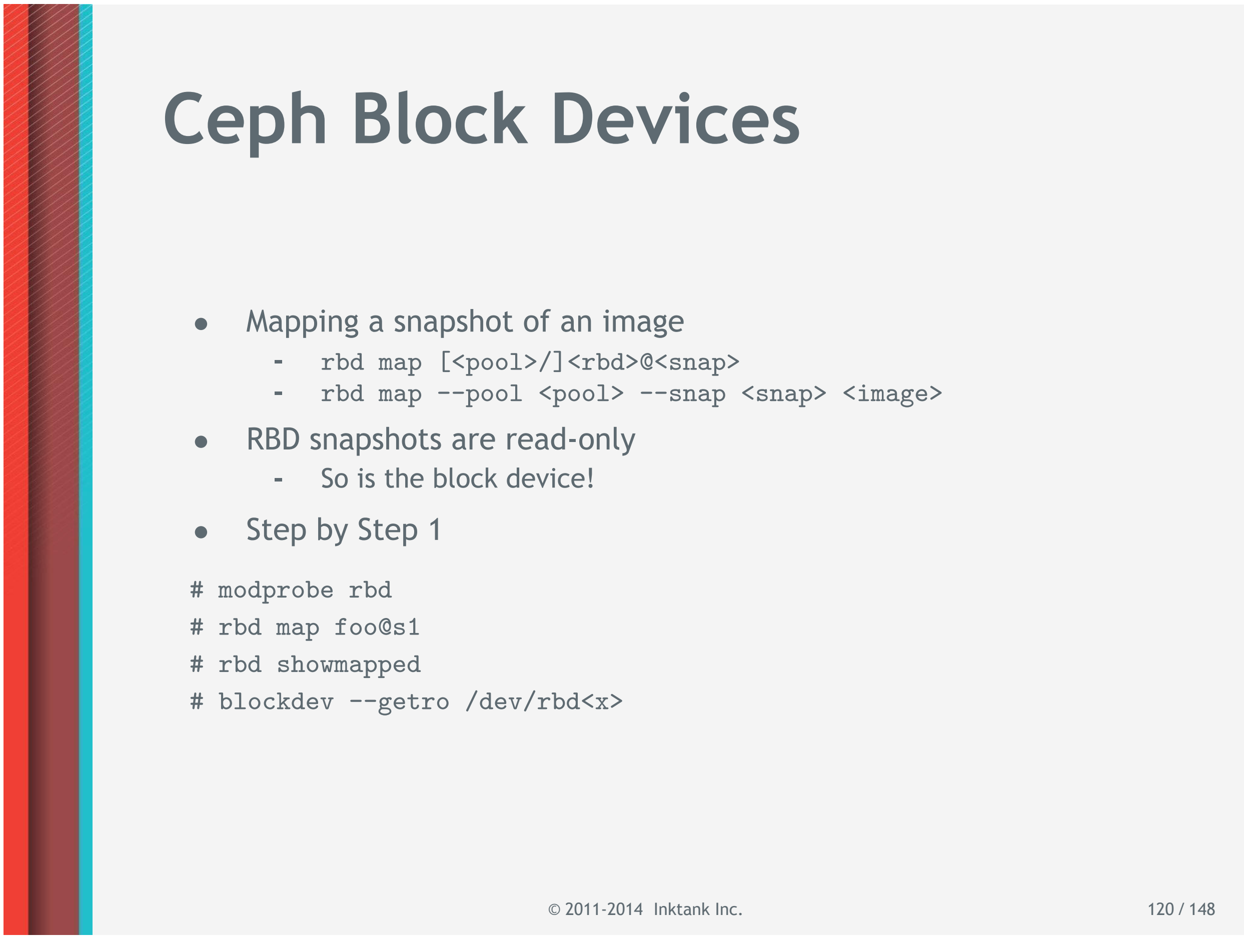
Ceph Block Devices

- Step by step
  - Mapping an RBD requires that the kernel module is loaded
    - `modprobe rbd` output should be silent
- Map the desired RBD to the client
  - `rbd map <rbd name>` output should be silent
  - If cephx authentication is enabled
    - Add the --id <name> and --keyring <filename> options
  - Checking the RBD device mapping
    - `rbd showmapped` [1]
  - Mount the RBD device

```
# modprobe rbd
# mkdir /mnt/mountpoint
# mkfs.ext4 /dev/rbd<x>
# mount /dev/rbdx /mnt/mountpoint
```

Note 1 : Verifying the RBD image is mapped to the client

```
# modprobe rbd
# rbd map foo
# rbd showmapped
```



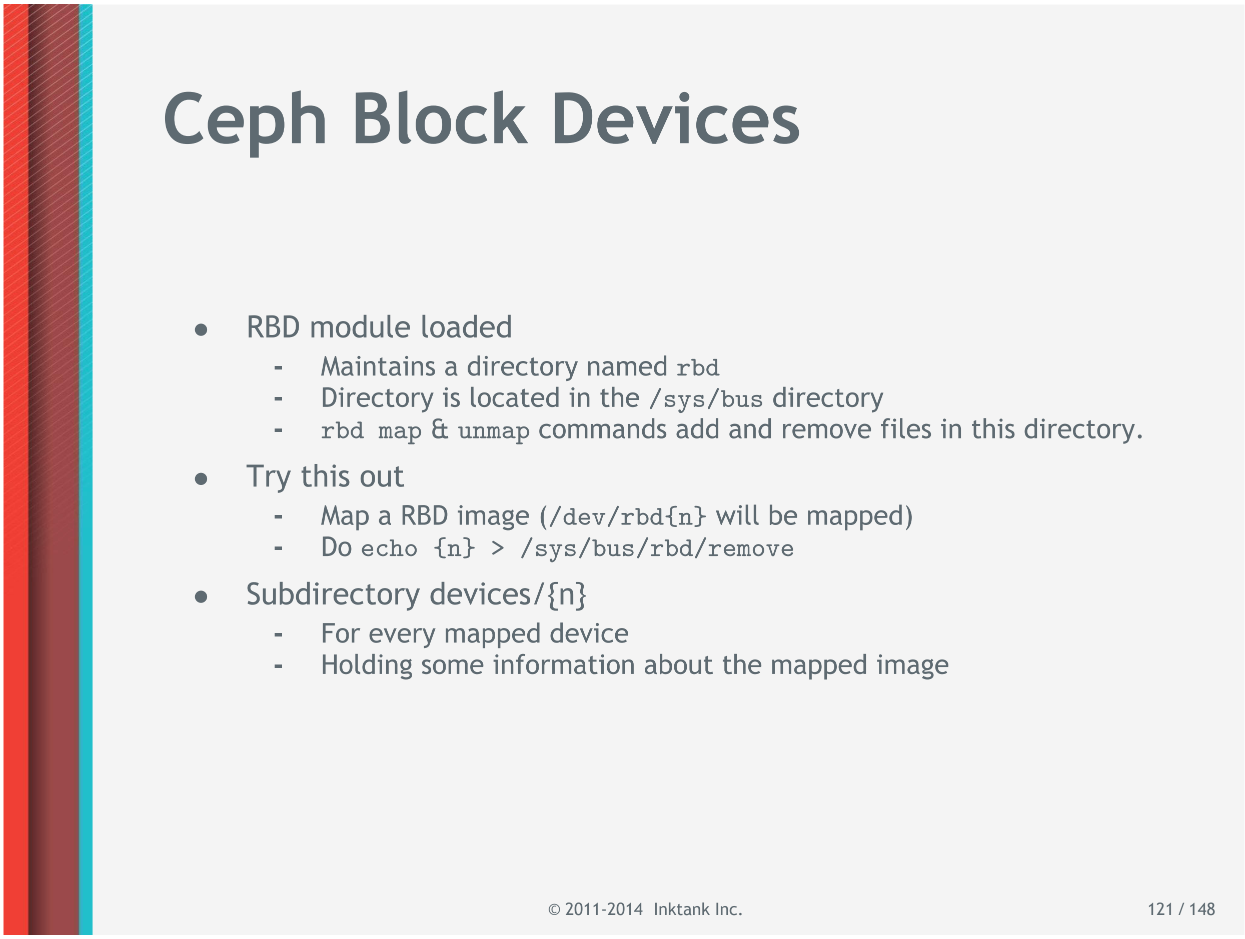
Ceph Block Devices

- Mapping a snapshot of an image
  - `rbd map [<pool>/]<rbd>@<snap>`
  - `rbd map --pool <pool> --snap <snap> <image>`
- RBD snapshots are read-only
  - So is the block device!
- Step by Step 1

```
# modprobe rbd
# rbd map foo@s1
# rbd showmapped
# blockdev --getro /dev/rbd<x>
```

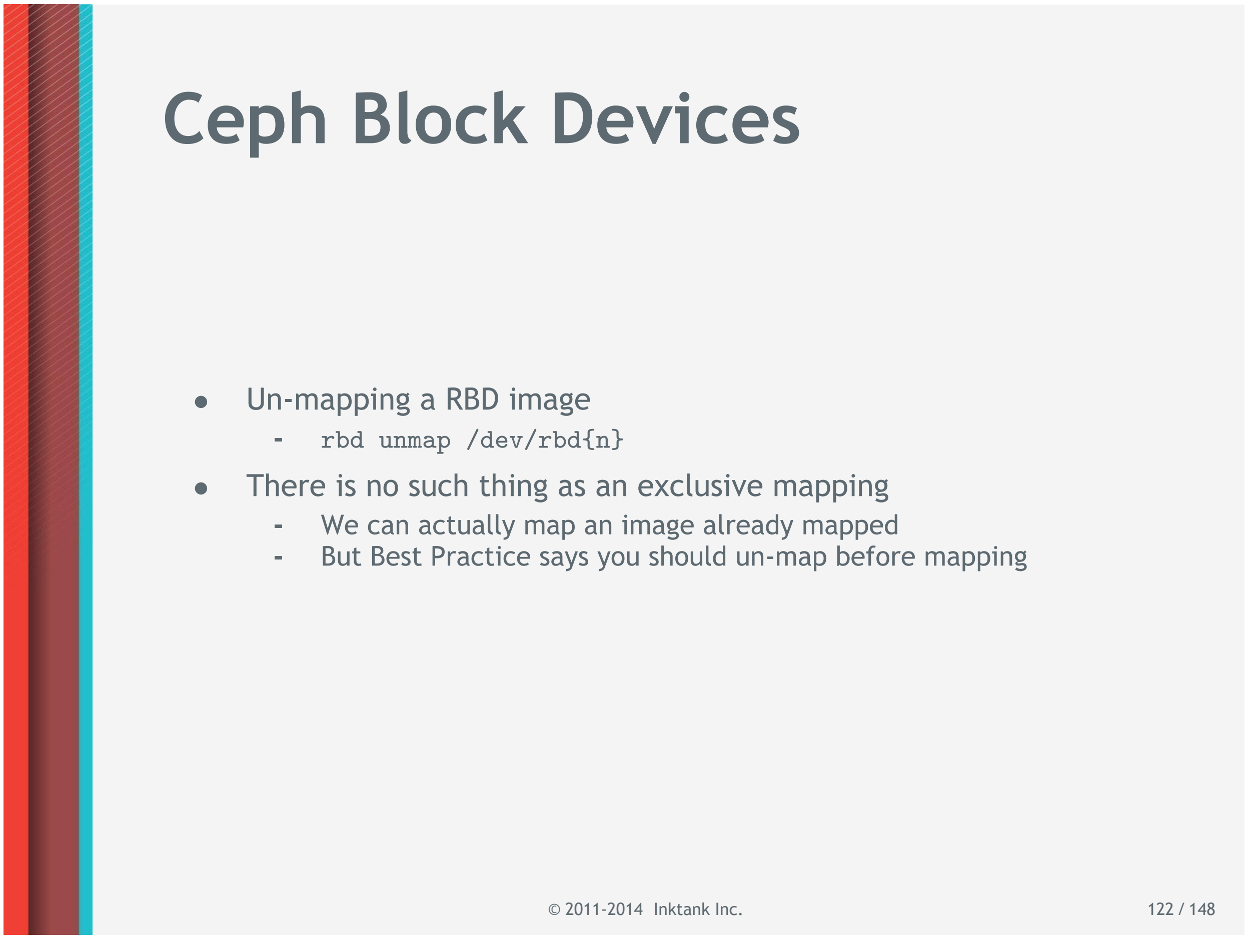
Note 1 : How to use a RBD snapshot on a client

```
  # modprobe rbd
  # rbd map foo@s1
  # rbd showmapped
  # blockdev --getro /dev/rbd1
  1
```



Ceph Block Devices

- RBD module loaded
  - Maintains a directory named `rbd`
  - Directory is located in the `/sys/bus` directory
  - `rbd map` & `unmap` commands add and remove files in this directory.
- Try this out
  - Map a RBD image (`/dev/rbd{n}` will be mapped)
  - Do `echo {n} > /sys/bus/rbd/remove`
- Subdirectory devices/{n}
  - For every mapped device
  - Holding some information about the mapped image

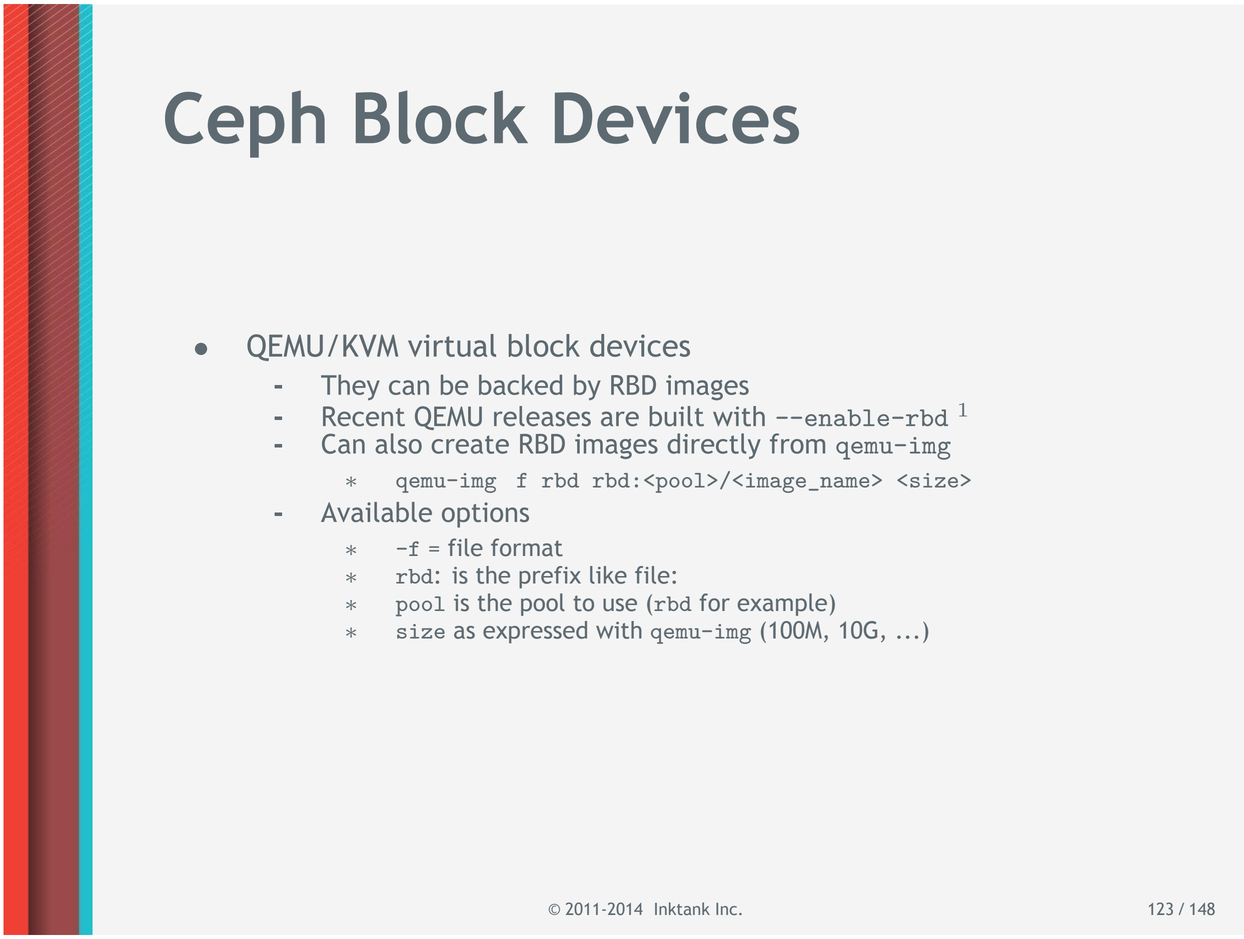


Ceph Block Devices

- Un-mapping a RBD image
  - `rbd unmap /dev/rbd{n}`
- There is no such thing as an exclusive mapping
  - We can actually map an image already mapped
  - But Best Practice says you should un-map before mapping

Remember the `pool` option can be replaced with the `p` option for quicker typing

```
  # rbd unmap /dev/rbd0
  # rbd showmapped
```

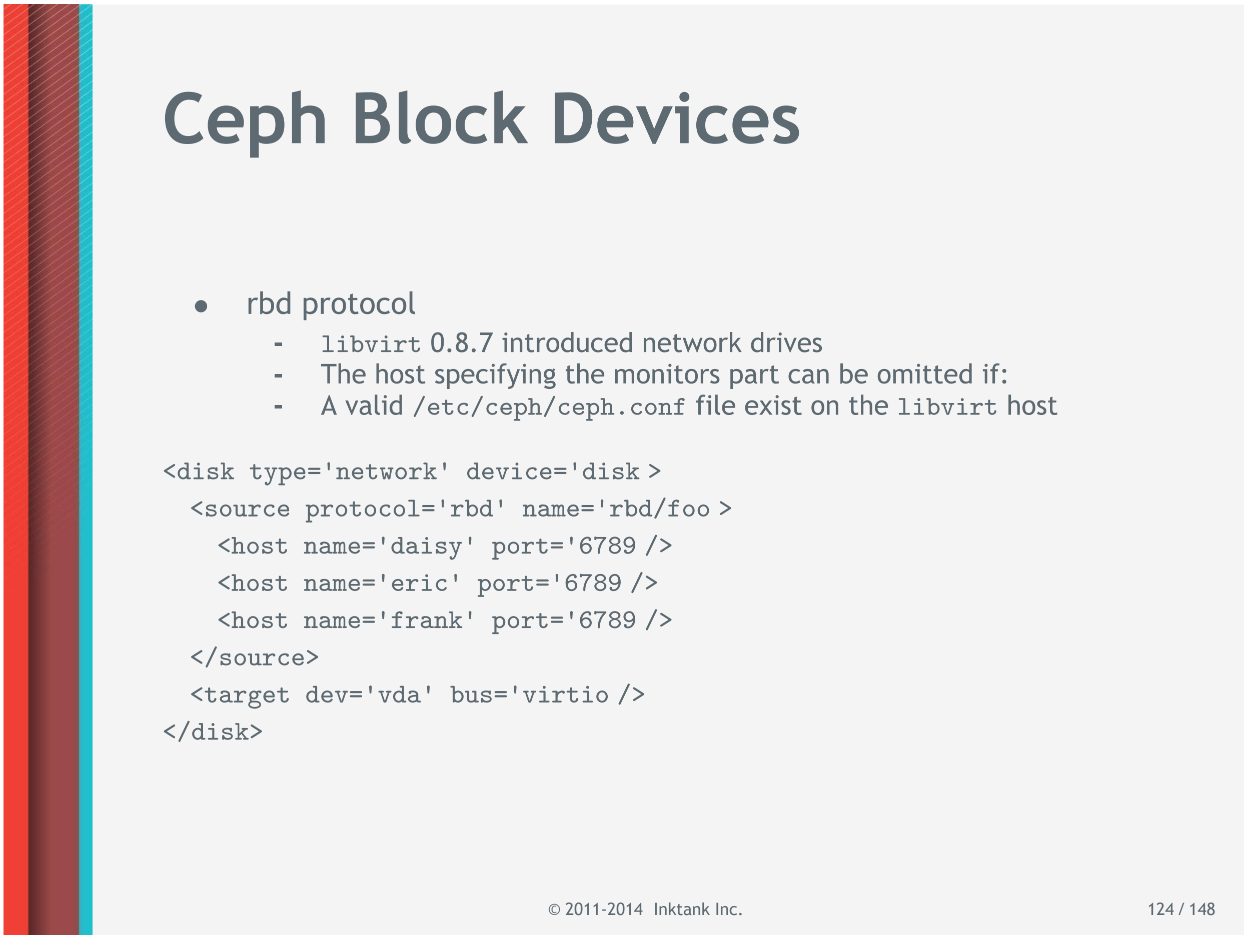


Ceph Block Devices

- QEMU/KVM virtual block devices
  - They can be backed by RBD images
  - Recent QEMU releases are built with `--enable-rbd` [1]
  - Can also create RBD images directly from `qemu-img`
    - `qemu-img  f rbd rbd:<pool>/<image_name> <size>`
  - Available options
    - `-f` = file format
    - `rbd:` is the prefix like file:
    - `pool` is the pool to use (`rbd` for example)
    - `size` as expressed with `qemu-img` (100M, 10G, ...)

Note 1 : e.g. qemu-utils packaged with Ubuntu 12.04

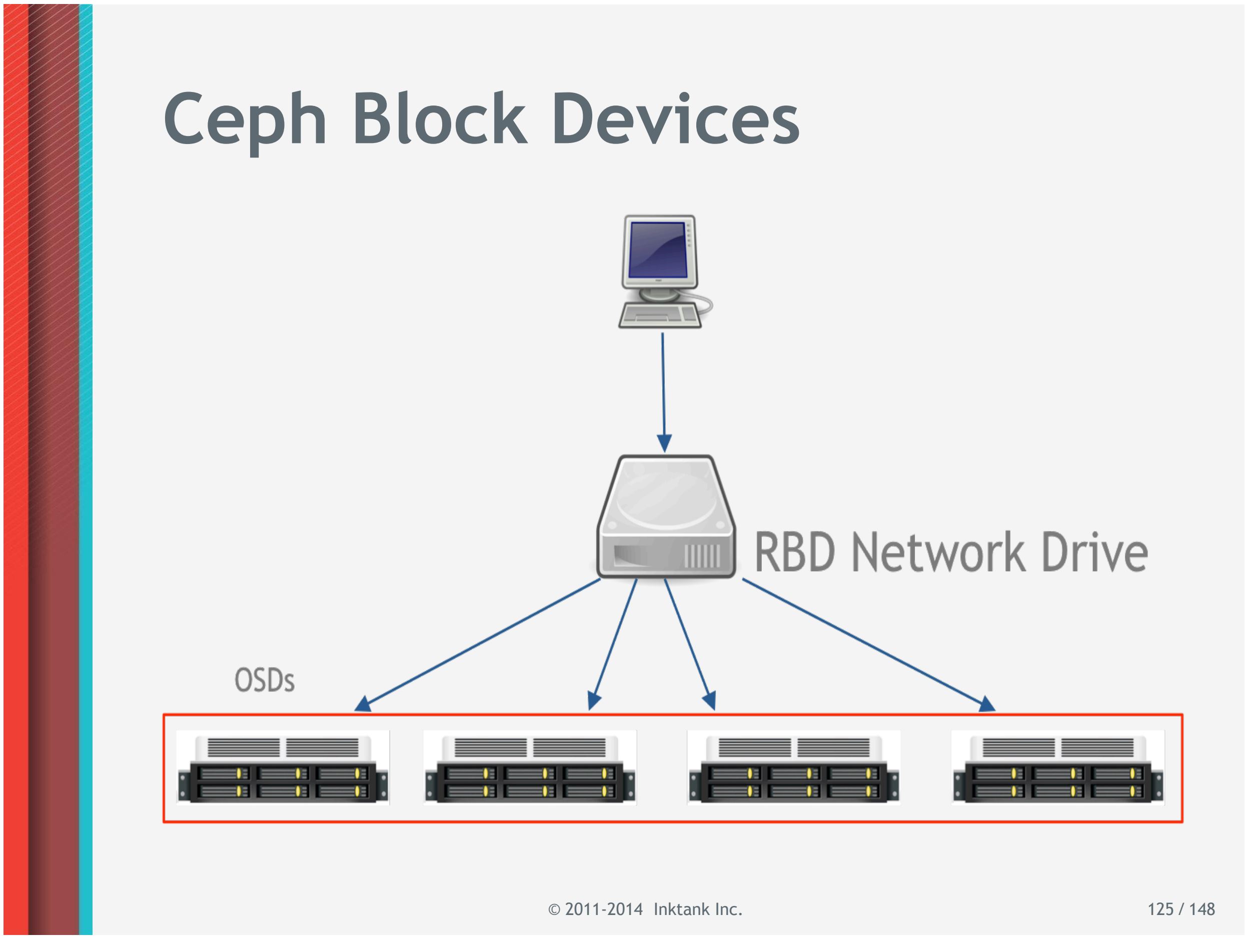
```
  # qemu-img create -f rbd rbd:rbd/qemu 1G
```



Ceph Block Devices

- rbd protocol
  - `libvirt` 0.8.7 introduced network drives
  - The host specifying the monitors part can be omitted if:
  - A valid `/etc/ceph/ceph.conf` file exist on the `libvirt` host

```
<disk type='network' device='disk >
  <source protocol='rbd' name='rbd/foo >
    <host name='daisy' port='6789 />
    <host name='eric' port='6789 />
    <host name='frank' port='6789 />
  </source>
  <target dev='vda' bus='virtio />
</disk>
```



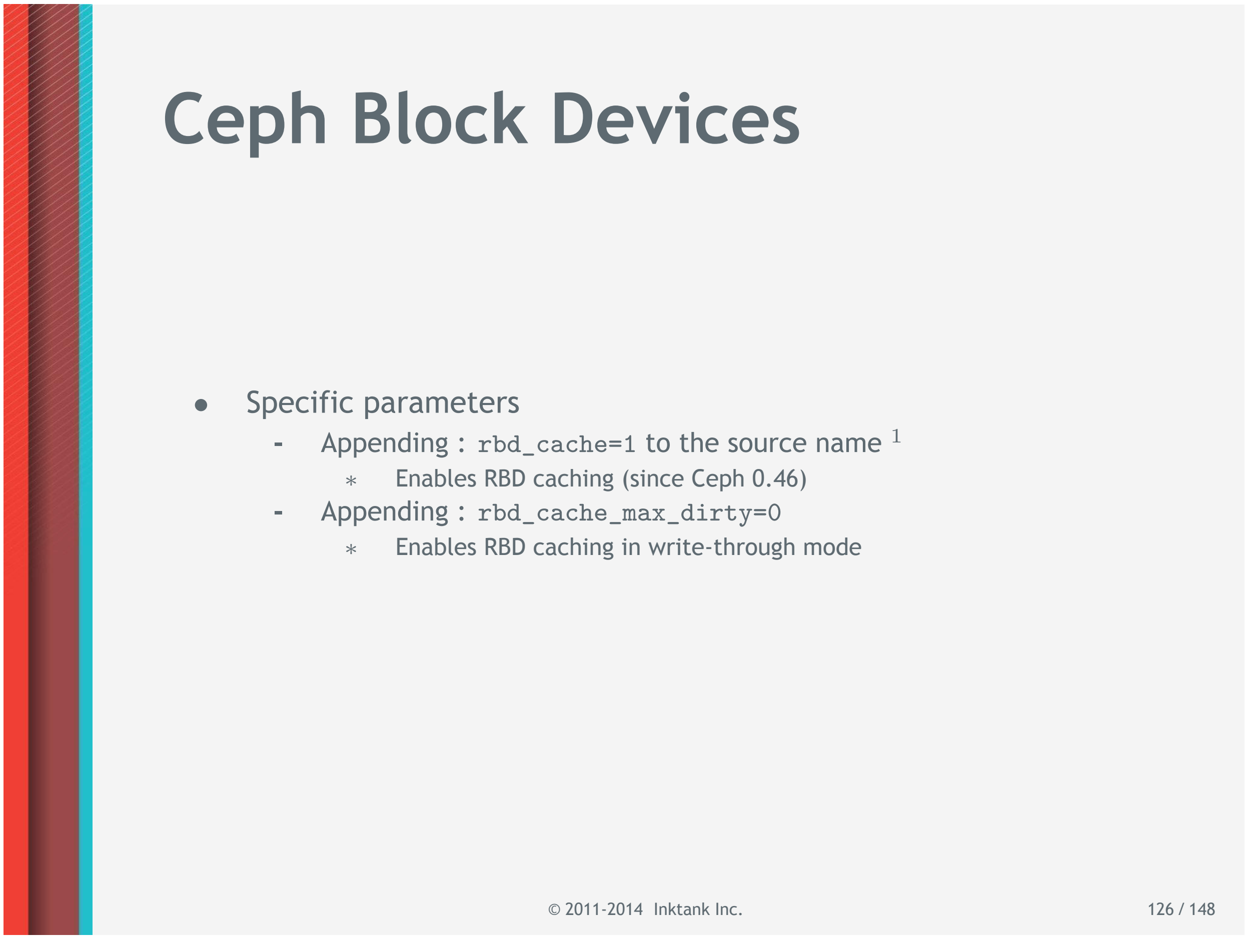
Ceph Block Devices

RBD Network Drive

OSDs

125 / 148

As you can see in this slide, data coming from a client to an RBD image will be split among the various OSD processes and underlying drives. As explained on the previous slide.

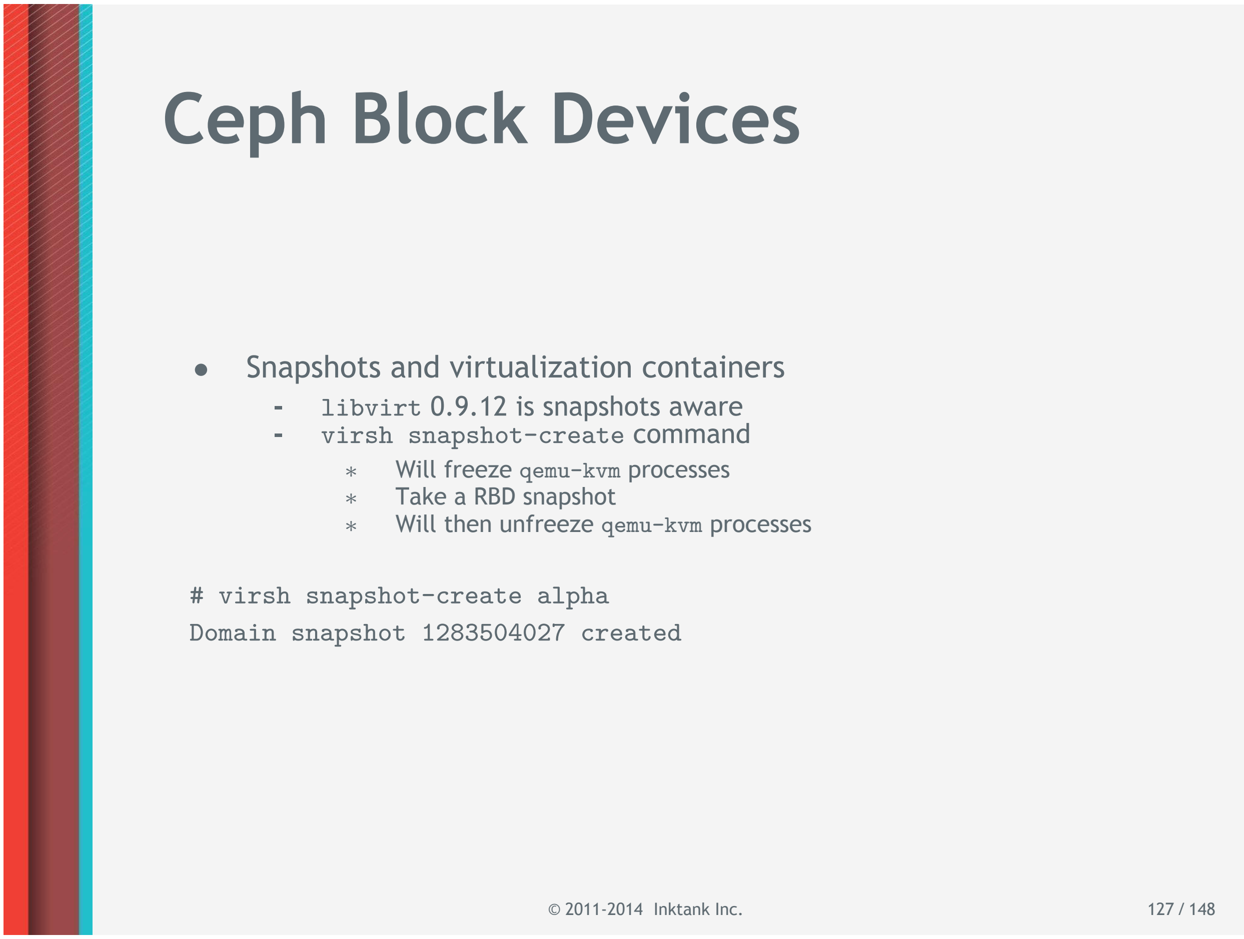


Ceph Block Devices

- Specific parameters
  - Appending : `rbd_cache=1` to the source name [1]
    - Enables RBD caching (since Ceph 0.46)
  - Appending : `rbd_cache_max_dirty=0`
    - Enables RBD caching in write-through mode

Note 1 : `rbd_cache=true` example

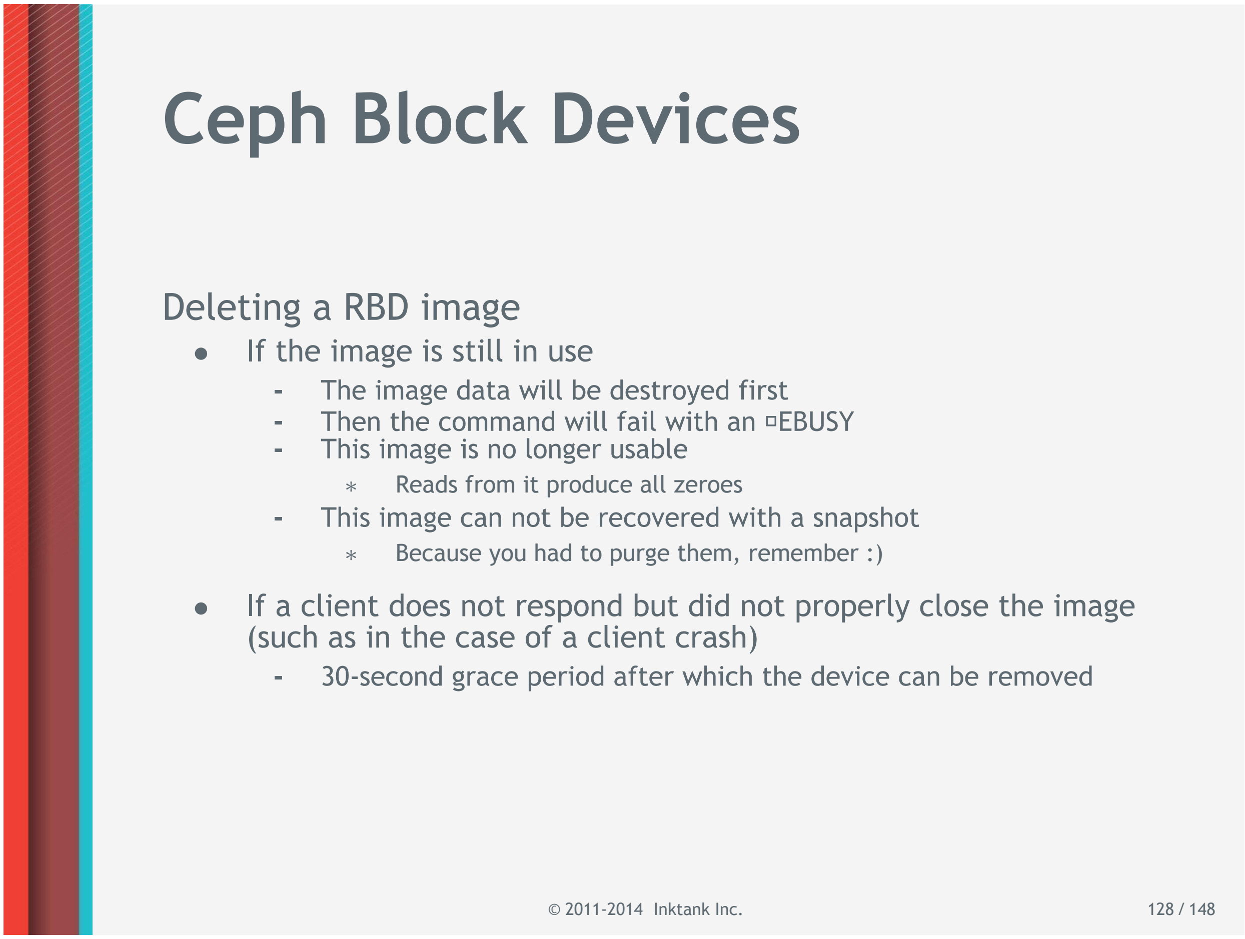
```
<disk type='network' device='disk'>
  <source protocol='rbd'name='rbd/foo:rbd_cache=1'>
    <host name='daisy' port='6789'>
    <host name='eric' port='6789'>
    <host name='frank' port='6789'>
  </source>
  <target dev='vda' bus='virtio'>
</disk>
```



Ceph Block Devices

- Snapshots and virtualization containers
  - `libvirt` 0.9.12 is snapshots aware
  - `virsh snapshot-create` command
    - Will freeze `qemu-kvm` processes
    - Take a RBD snapshot
    - Will then unfreeze `qemu-kvm` processes

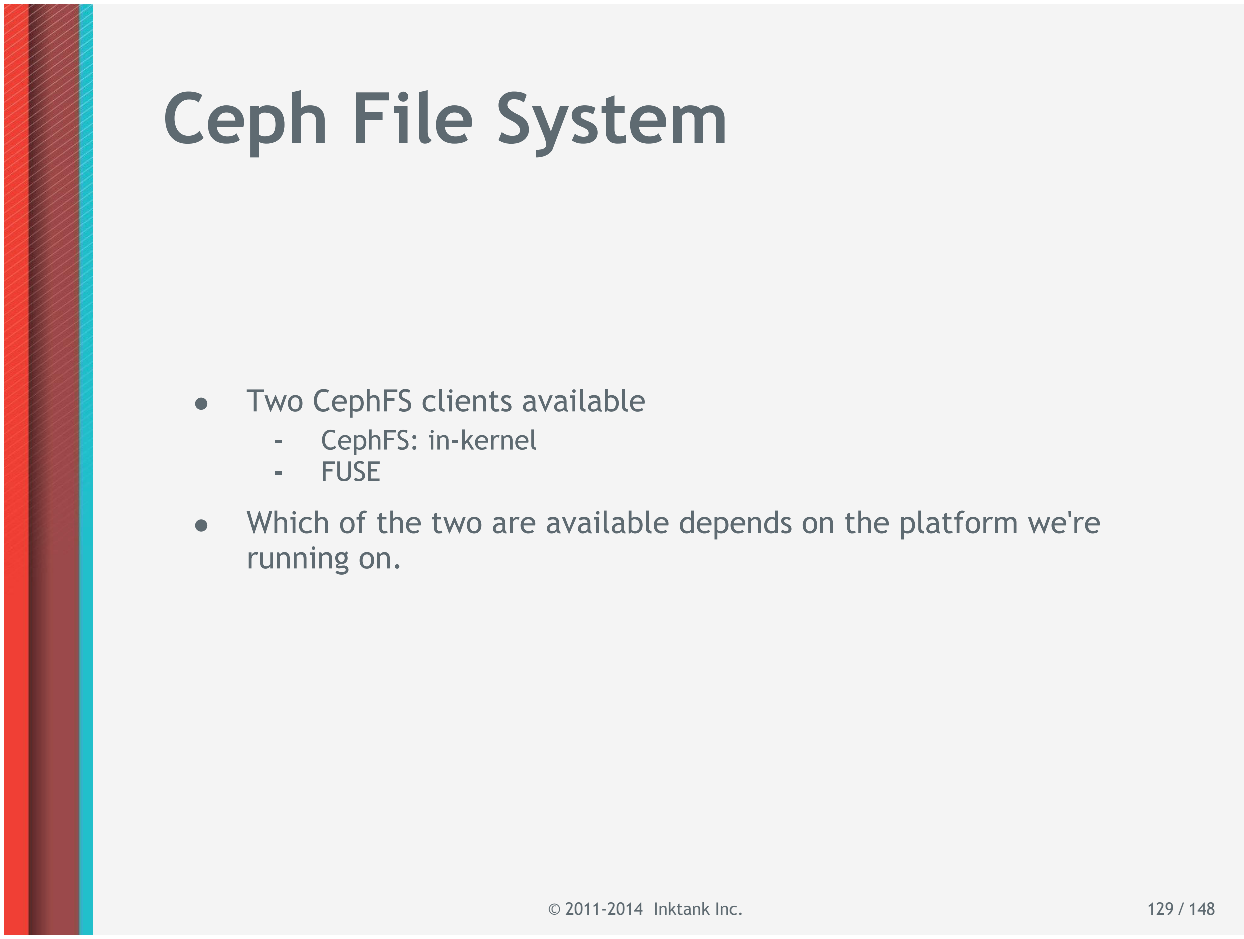
```
# virsh snapshot-create alpha
Domain snapshot 1283504027 created
```



Ceph Block Devices

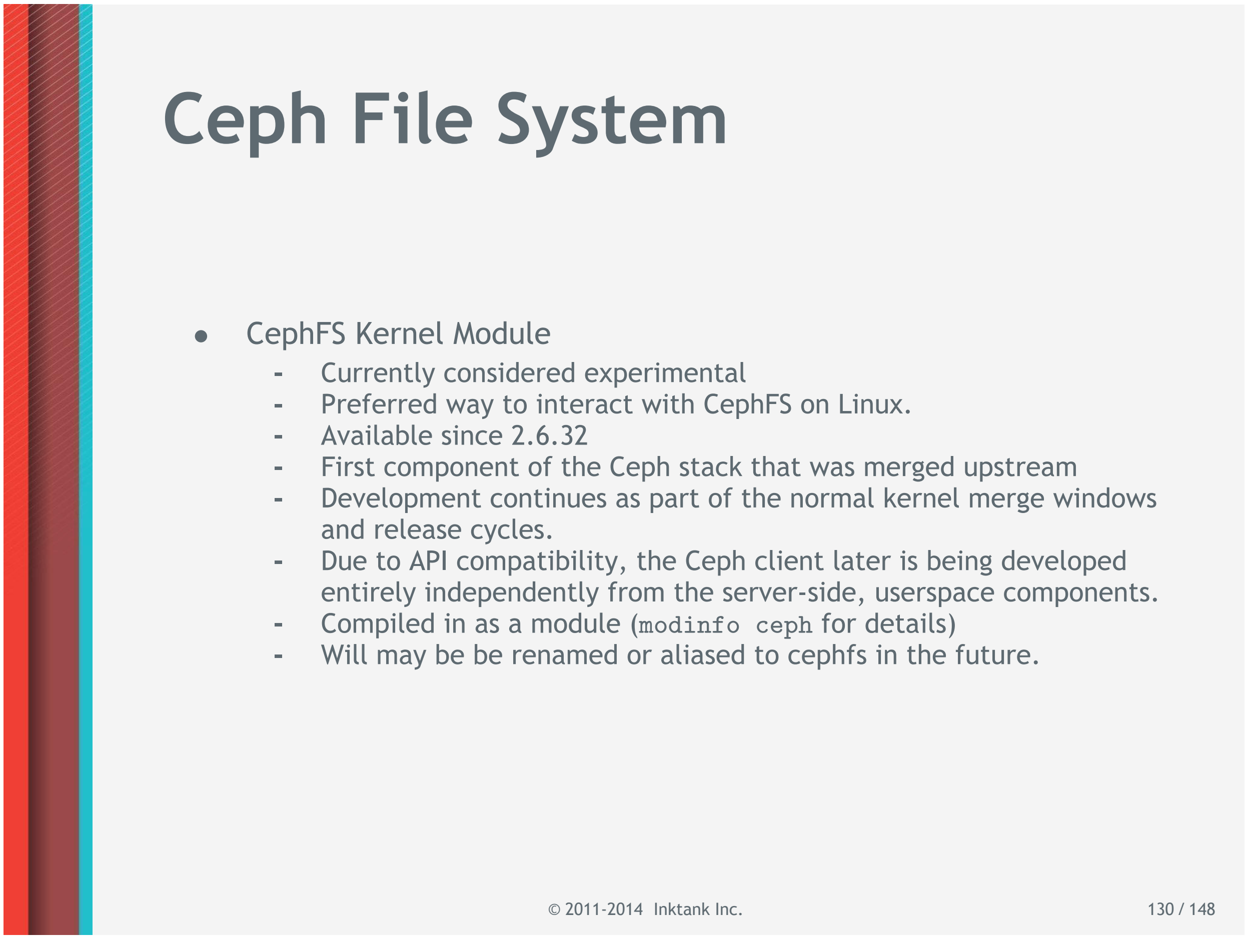
Deleting a RBD image

- If the image is still in use
  - The image data will be destroyed first
  - Then the command will fail with an ¤EBUSY
  - This image is no longer usable
    - Reads from it produce all zeroes
  - This image can not be recovered with a snapshot
    - Because you had to purge them, remember :)
- If a client does not respond but did not properly close the image (such as in the case of a client crash)
  - 30-second grace period after which the device can be removed



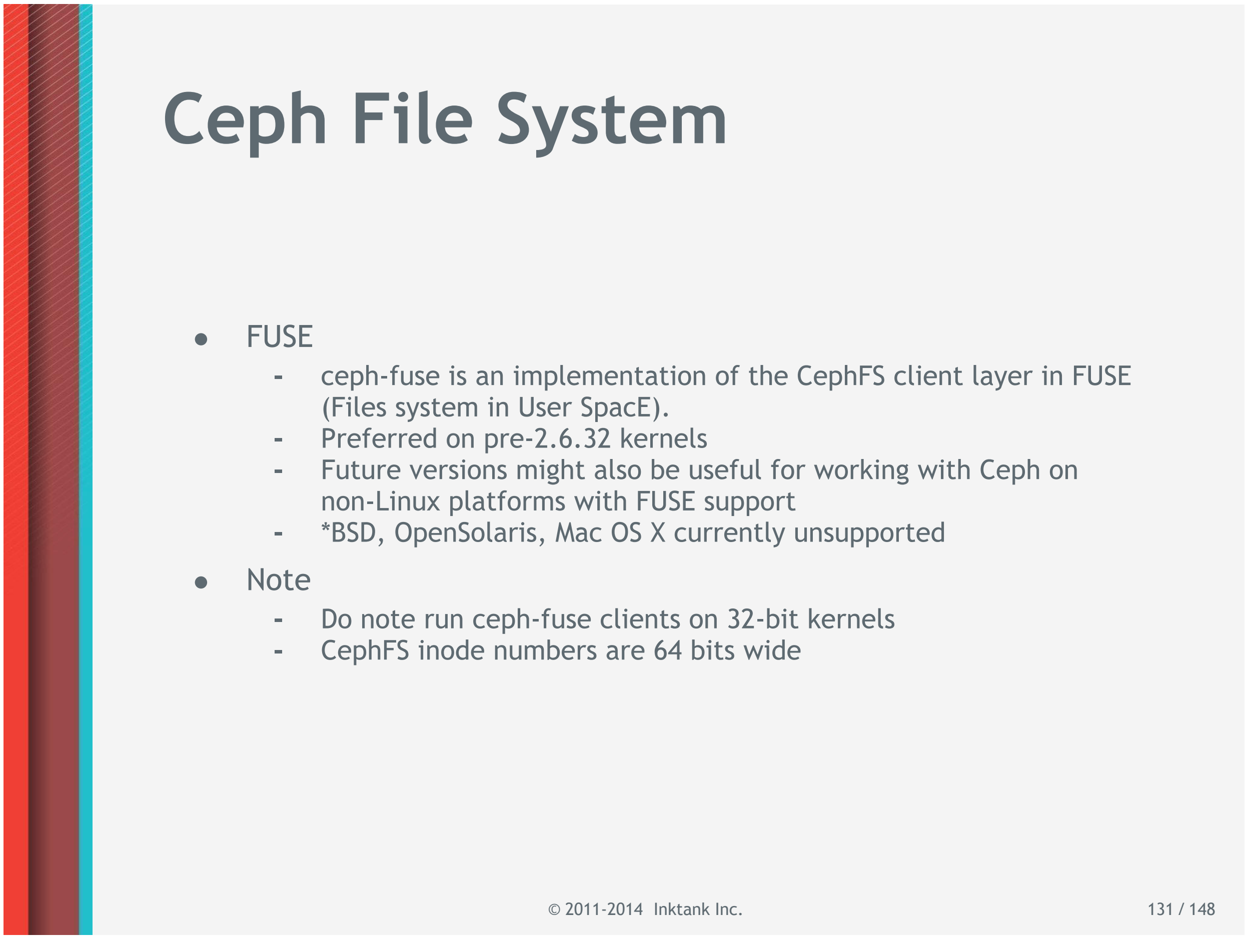
Ceph File System

- Two CephFS clients available
  - CephFS: in-kernel
  - FUSE
- Which of the two are available depends on the platform we're running on.



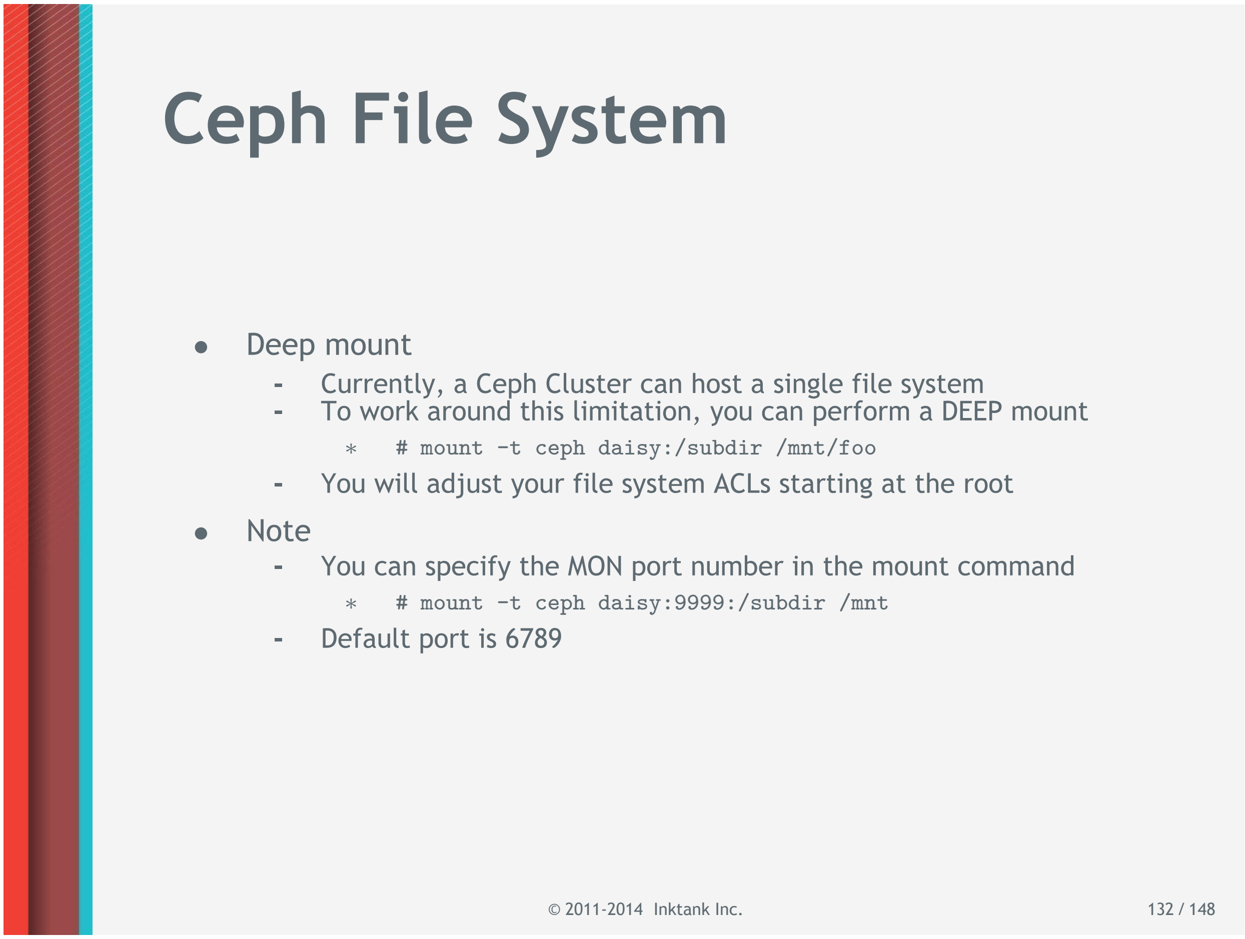
Ceph File System

- CephFS Kernel Module
  - Currently considered experimental
  - Preferred way to interact with CephFS on Linux.
  - Available since 2.6.32
  - First component of the Ceph stack that was merged upstream
  - Development continues as part of the normal kernel merge windows and release cycles.
  - Due to API compatibility, the Ceph client later is being developed entirely independently from the server-side, userspace components.
  - Compiled in as a module (`modinfo ceph` for details)
  - Will may be be renamed or aliased to cephfs in the future.



Ceph File System

- FUSE
  - ceph-fuse is an implementation of the CephFS client layer in FUSE (Files system in User SpacE).
  - Preferred on pre-2.6.32 kernels
  - Future versions might also be useful for working with Ceph on non-Linux platforms with FUSE support
  - *BSD, OpenSolaris, Mac OS X currently unsupported
- Note
  - Do note run ceph-fuse clients on 32-bit kernels
  - CephFS inode numbers are 64 bits wide

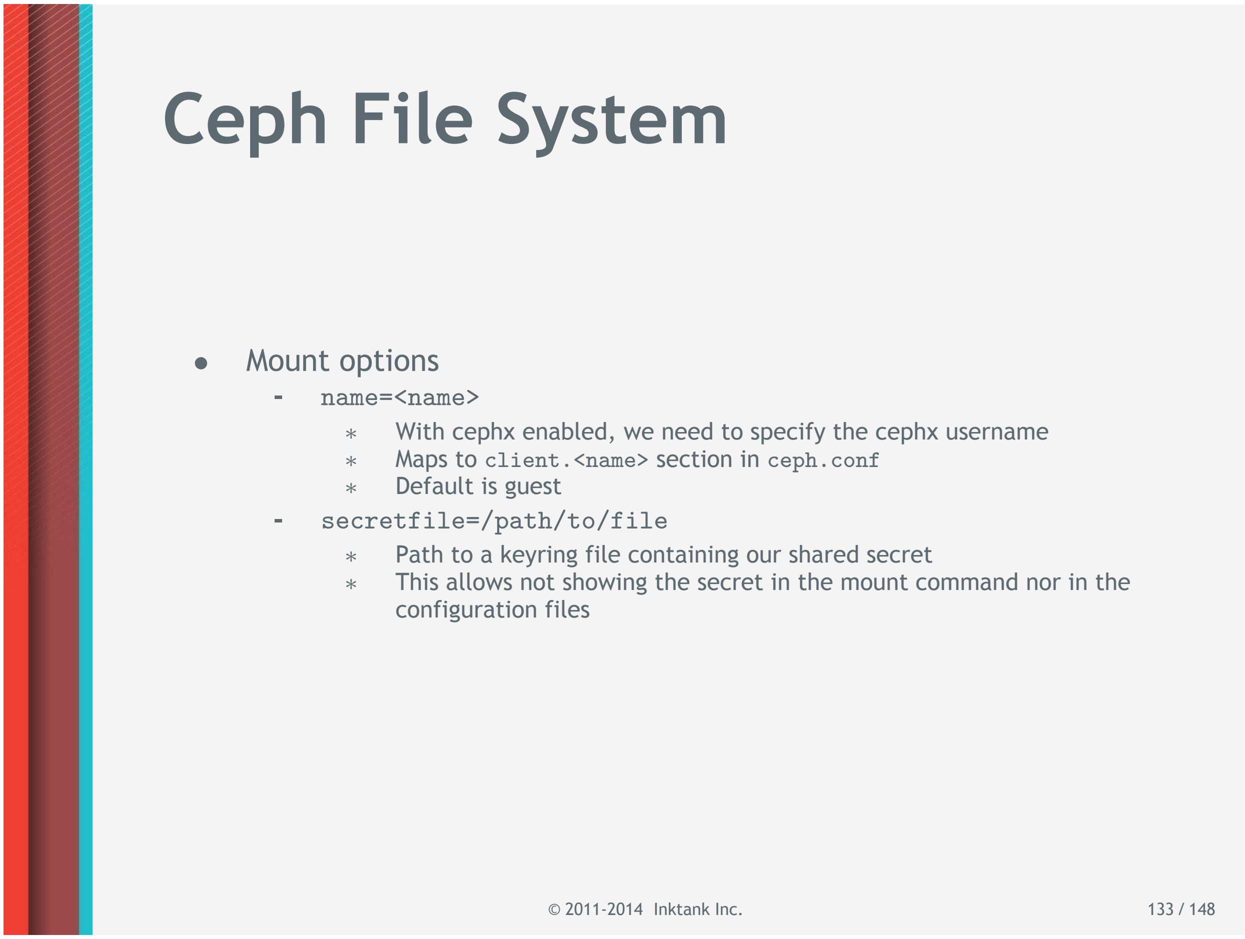


Ceph File System

- Deep mount
  - Currently, a Ceph Cluster can host a single file system
  - To work around this limitation, you can perform a DEEP mount

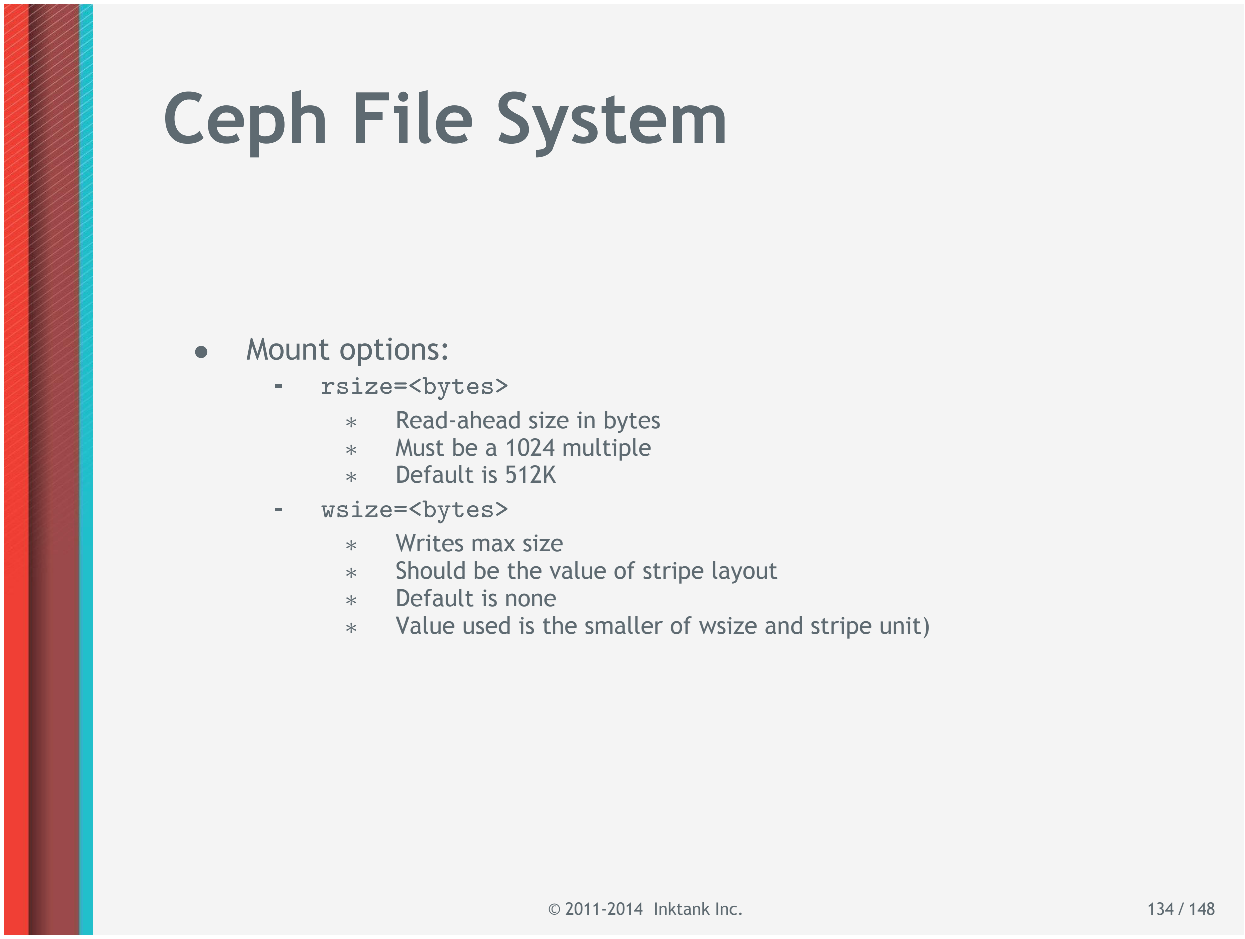
    -
      ```
      # mount -t ceph daisy:/subdir /mnt/foo
      ```
  - You will adjust your file system ACLs starting at the root
- Note
  - You can specify the MON port number in the mount command

    -
      ```
      # mount -t ceph daisy:9999:/subdir /mnt
      ```
  - Default port is 6789



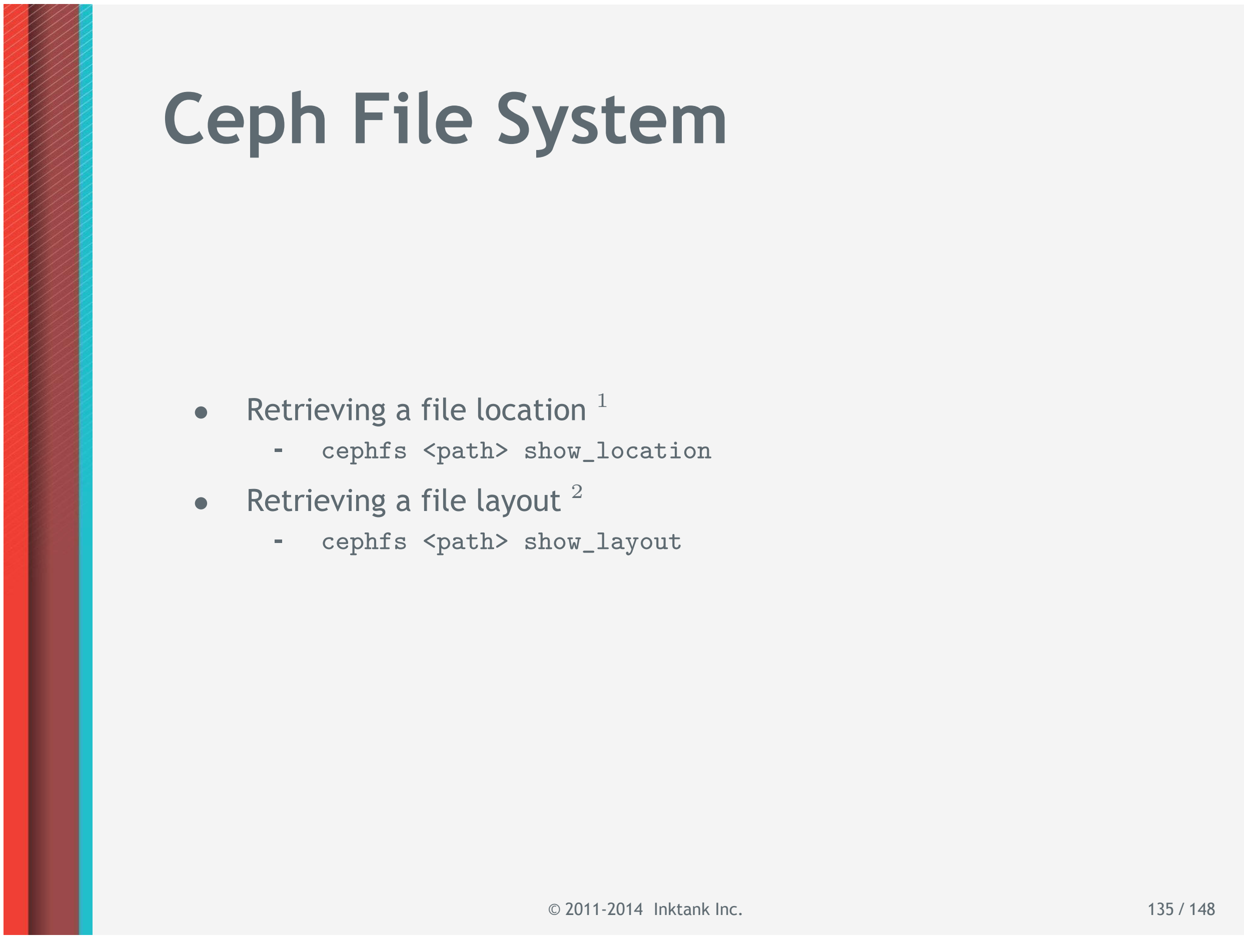
Ceph File System

- Mount options
  - `name=<name>`
    - * With cephx enabled, we need to specify the cephx username
    - * Maps to `client.<name>` section in `ceph.conf`
    - * Default is guest
  - `secretfile=/path/to/file`
    - * Path to a keyring file containing our shared secret
    - * This allows not showing the secret in the mount command nor in the configuration files



Ceph File System

- Mount options:
  - `rsize=<bytes>`
    - Read-ahead size in bytes
    - Must be a 1024 multiple
    - Default is 512K
  - `wsize=<bytes>`
    - Writes max size
    - Should be the value of stripe layout
    - Default is none
    - Value used is the smaller of wsize and stripe unit)



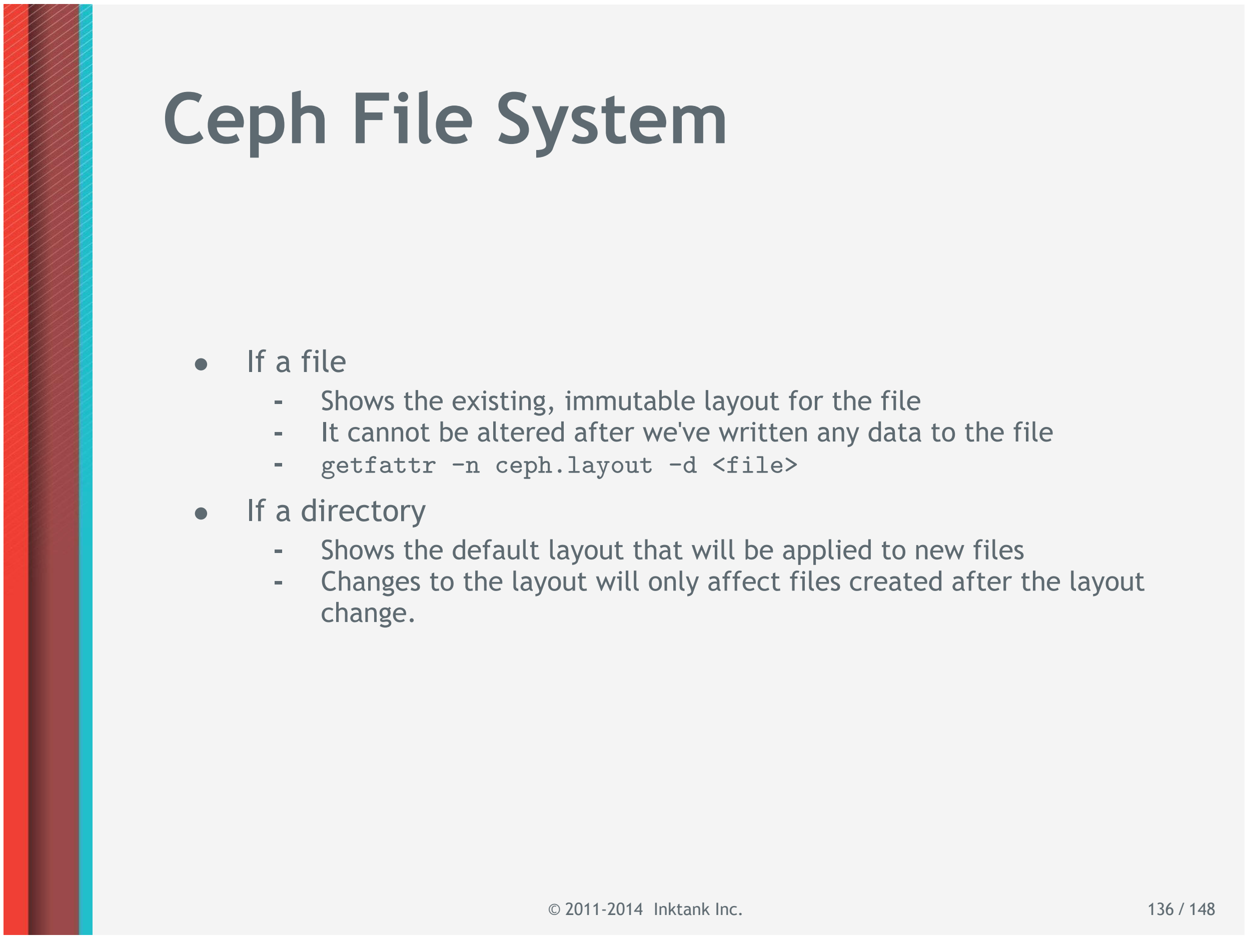
Ceph File System

- Retrieving a file location [1]
  - `cephfs <path> show_location`
- Retrieving a file layout [2]
  - `cephfs <path> show_layout`

Note 1 : The output in detail:
- `file_offset`: the offset (in bytes) from the start of the file
- `object_offset`: offset (bytes) from the start of the RADOS object
- `object_no`: the index among the RADOS objects mapped to the file. For offset 0, this will always be 0.
- `object_size`: the size of the RADOS object we're looking at, as per the defined layout.
- `object_name`: the RADOS object ID we're looking at (we can look up its more detailed OSD mapping with sdmaptool --test_map_object <object_name>)
- `block_offset`: the offset from the start of the stripe
- `block_size`: the stripe size, as per the defined layout.

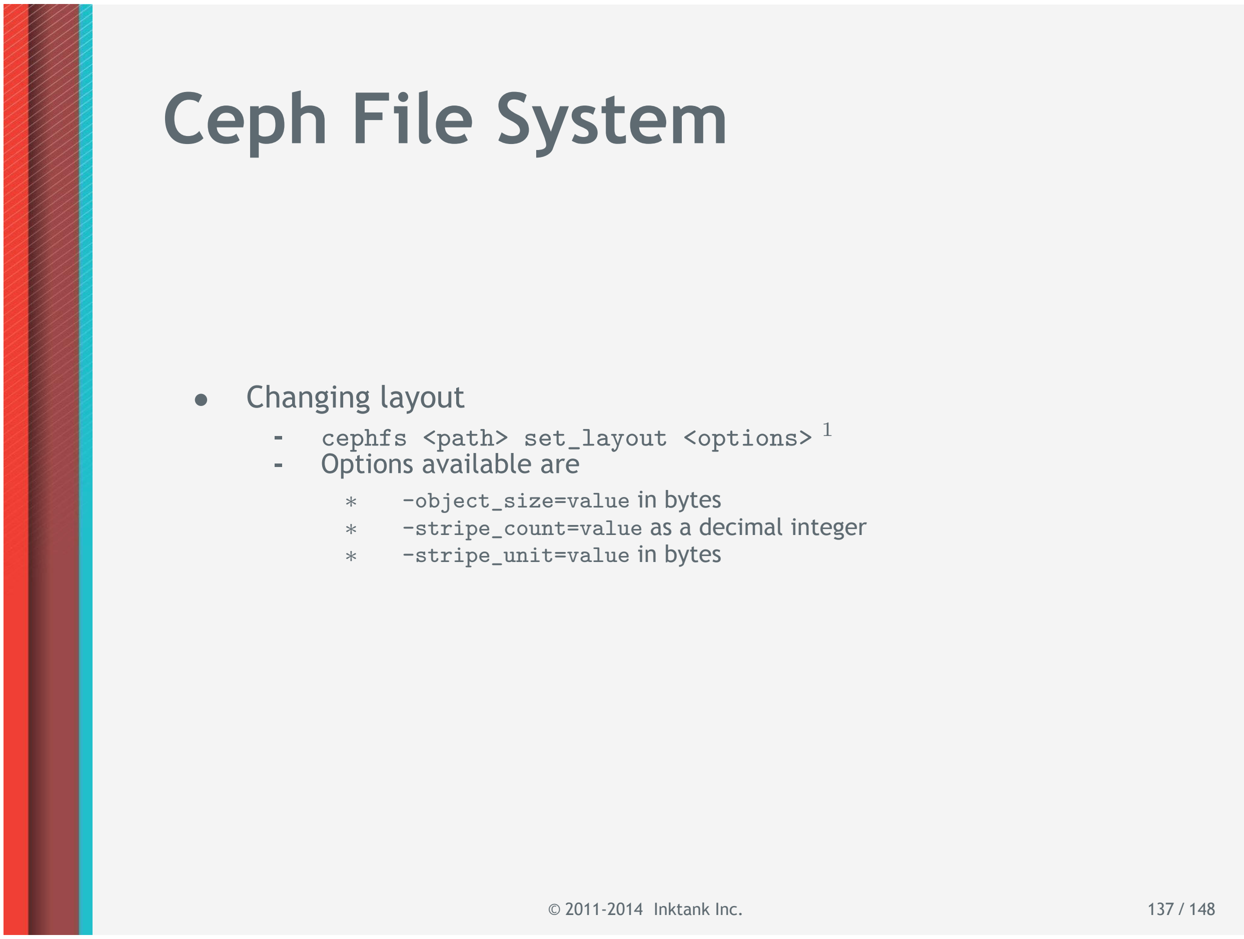
Note 2 : The output in detail:

```
- layout.data_pool:.....  0
- layout.object_size:...  4194304
- layout.stripe_unit:...  4194304
- layout.stripe_count:..  1
```



Ceph File System

- If a file
  - Shows the existing, immutable layout for the file
  - It cannot be altered after we've written any data to the file
  - `getfattr -n ceph.layout -d <file>`
- If a directory
  - Shows the default layout that will be applied to new files
  - Changes to the layout will only affect files created after the layout change.



Ceph File System

- Changing layout
  - `cephfs <path> set_layout <options>` [1]
  - Options available are
    - * `-object_size=value` in bytes
    - * `-stripe_count=value` as a decimal integer
    - * `-stripe_unit=value` in bytes

Note 1 : Syntax and parameters

```
# cephfs /mnt set_layout --object_size 4194304 --stripe_count 2 --stripe_unit $((4194304/2))
```

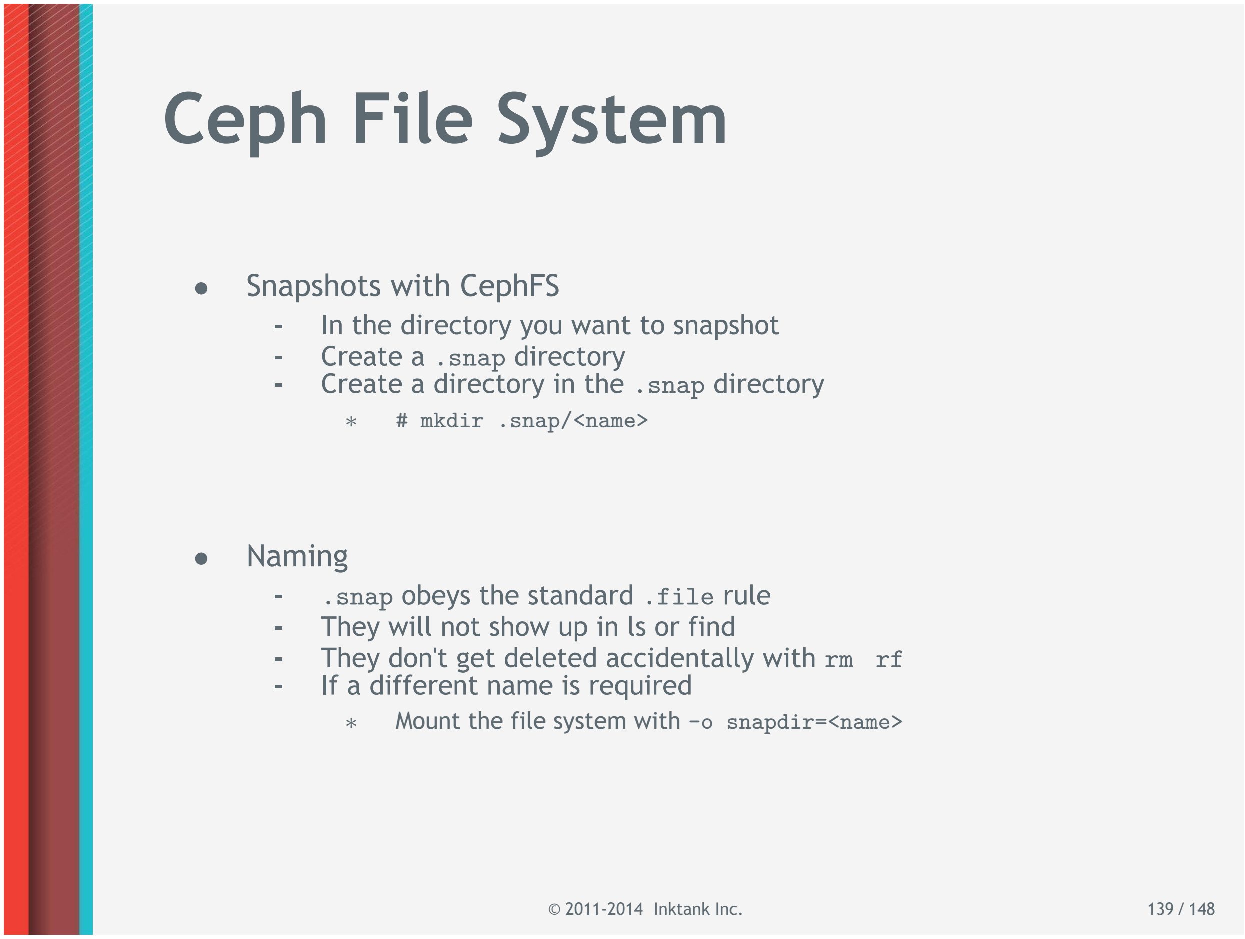
`--object_size` or `-s` sets the size of individual objects

`--stripe_count` or `-c` sets the number of stripes to distribute objects over

`--stripe_unit` or `-u` set the size of a stripe.

Ceph File System

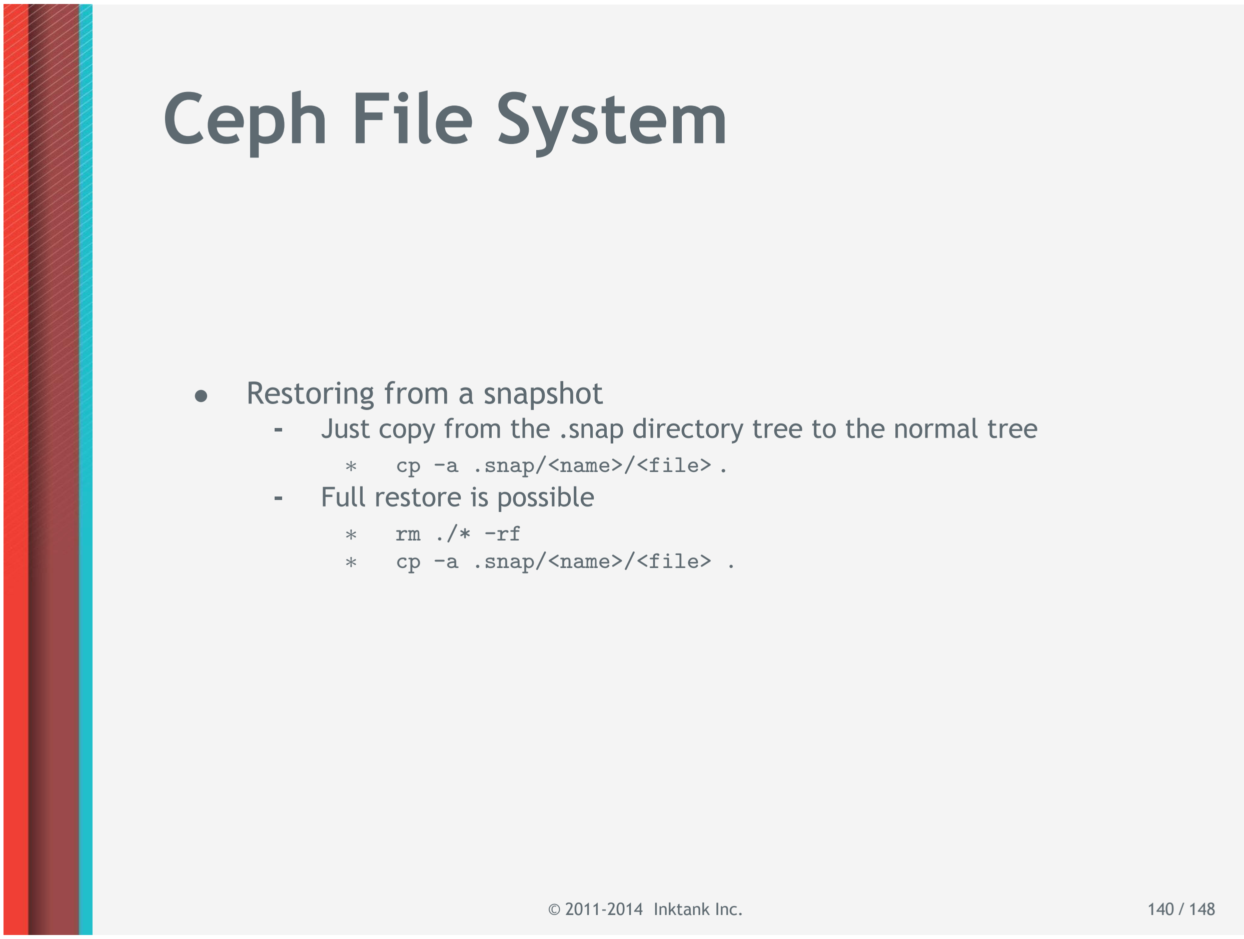
- Mount with FUSE
  - FUSE client reads /etc/ceph/ceph.conf
  - Only a mount point to specify
  - `# ceph-fuse /mnt`
- Un-mount with FUSE
  - `# ceph-fuse -u /mnt`



Ceph File System

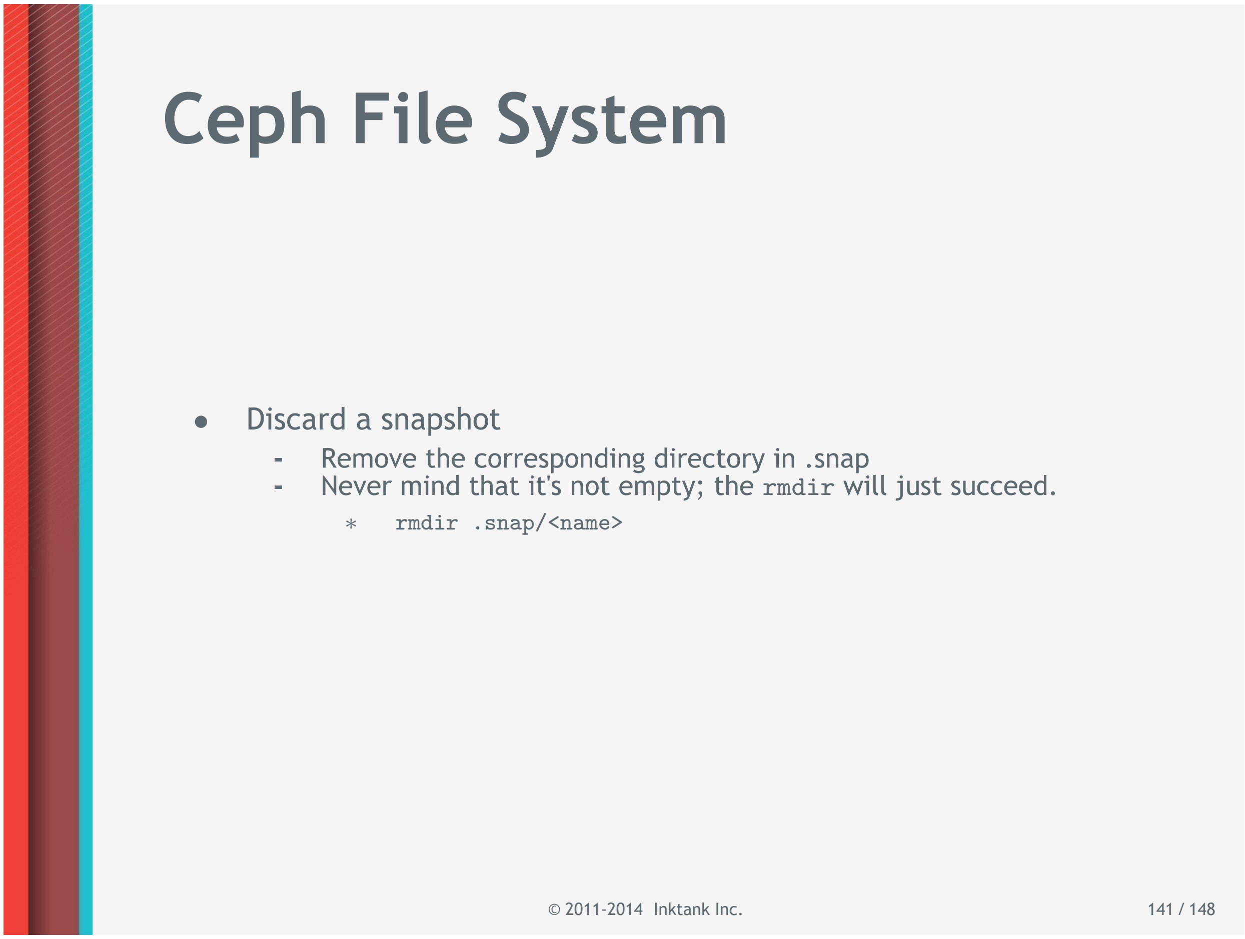
- Snapshots with CephFS
  - In the directory you want to snapshot
  - Create a `.snap` directory
  - Create a directory in the `.snap` directory
    - `# mkdir .snap/<name>`

- Naming
  - `.snap` obeys the standard `.file` rule
  - They will not show up in ls or find
  - They don't get deleted accidentally with `rm  rf`
  - If a different name is required
    - Mount the file system with `-o snapdir=<name>`



Ceph File System

- Restoring from a snapshot
  - Just copy from the .snap directory tree to the normal tree
    - `cp -a .snap/<name>/<file> .`
  - Full restore is possible
    - `rm ./* -rf`
    - `cp -a .snap/<name>/<file> .`

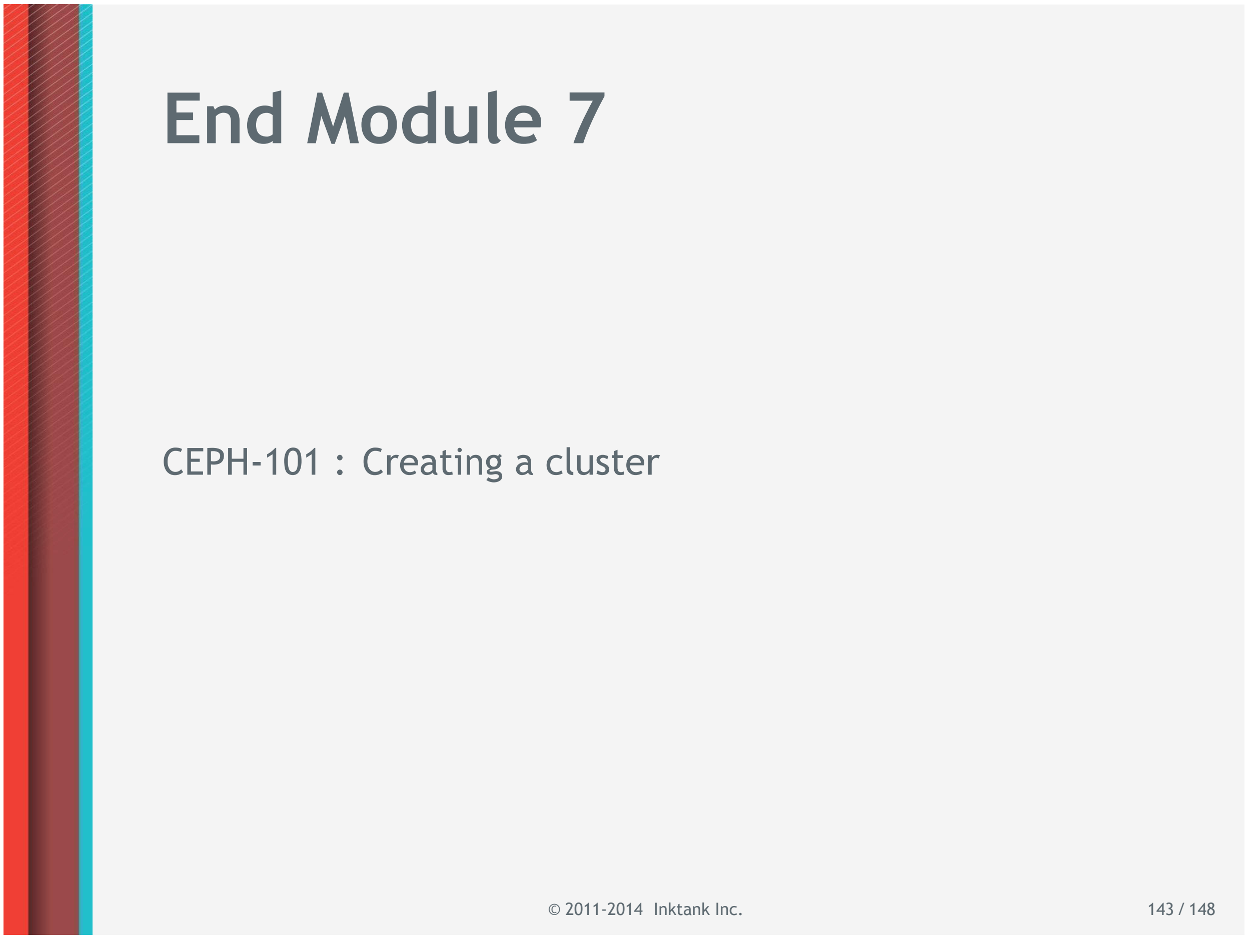


Ceph File System

- Discard a snapshot
  - Remove the corresponding directory in .snap
  - Never mind that it's not empty; the `rmdir` will just succeed.
    - * `rmdir .snap/<name>`

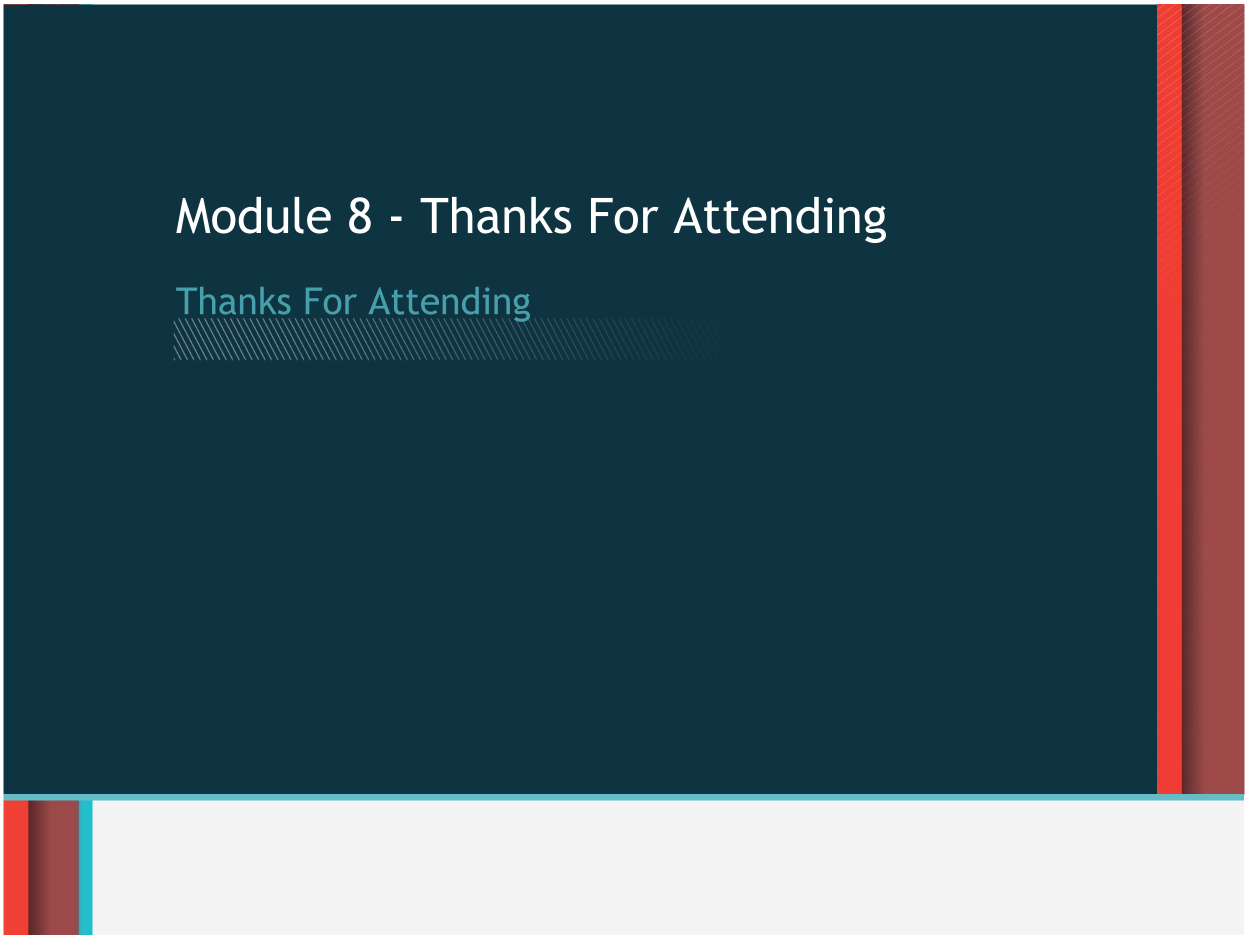
Summary

- Deploying a cluster
- Configuration file format
- Working with Ceph clients
  - `rados`
  - `rbd`
  - Mounting a CephFS File System



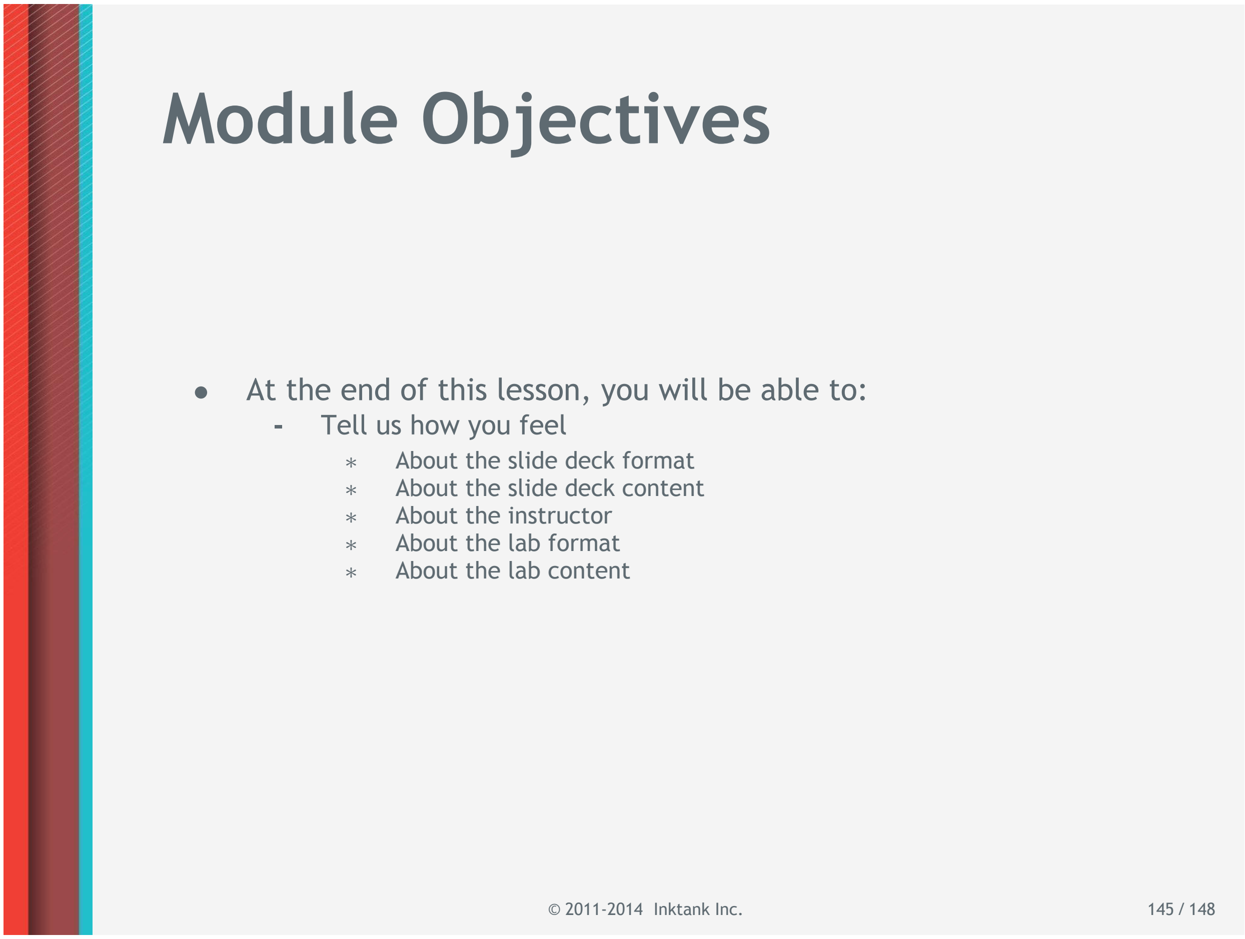
End Module 7

CEPH-101 : Creating a cluster



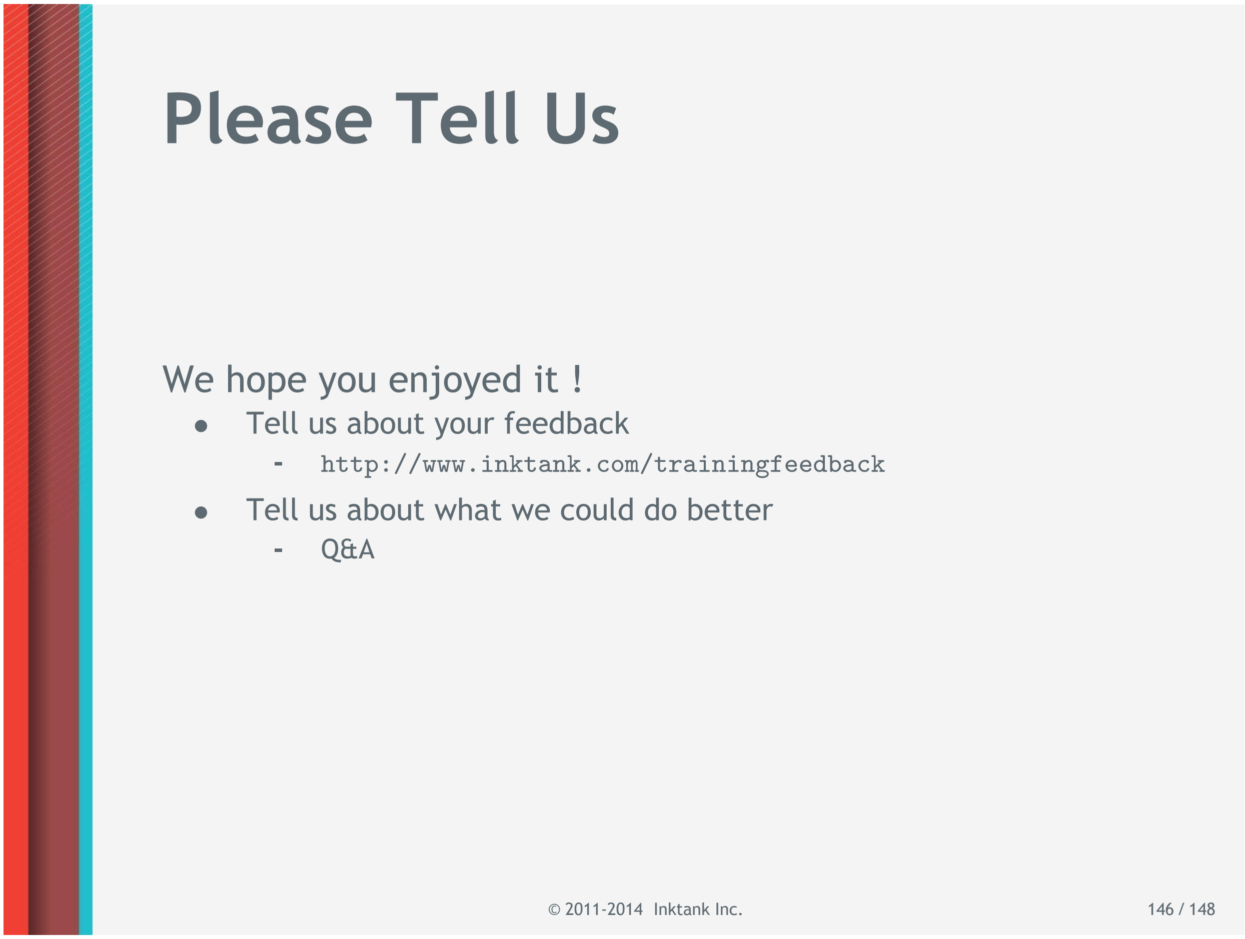
Module 8 - Thanks For Attending

Thanks For Attending



Module Objectives

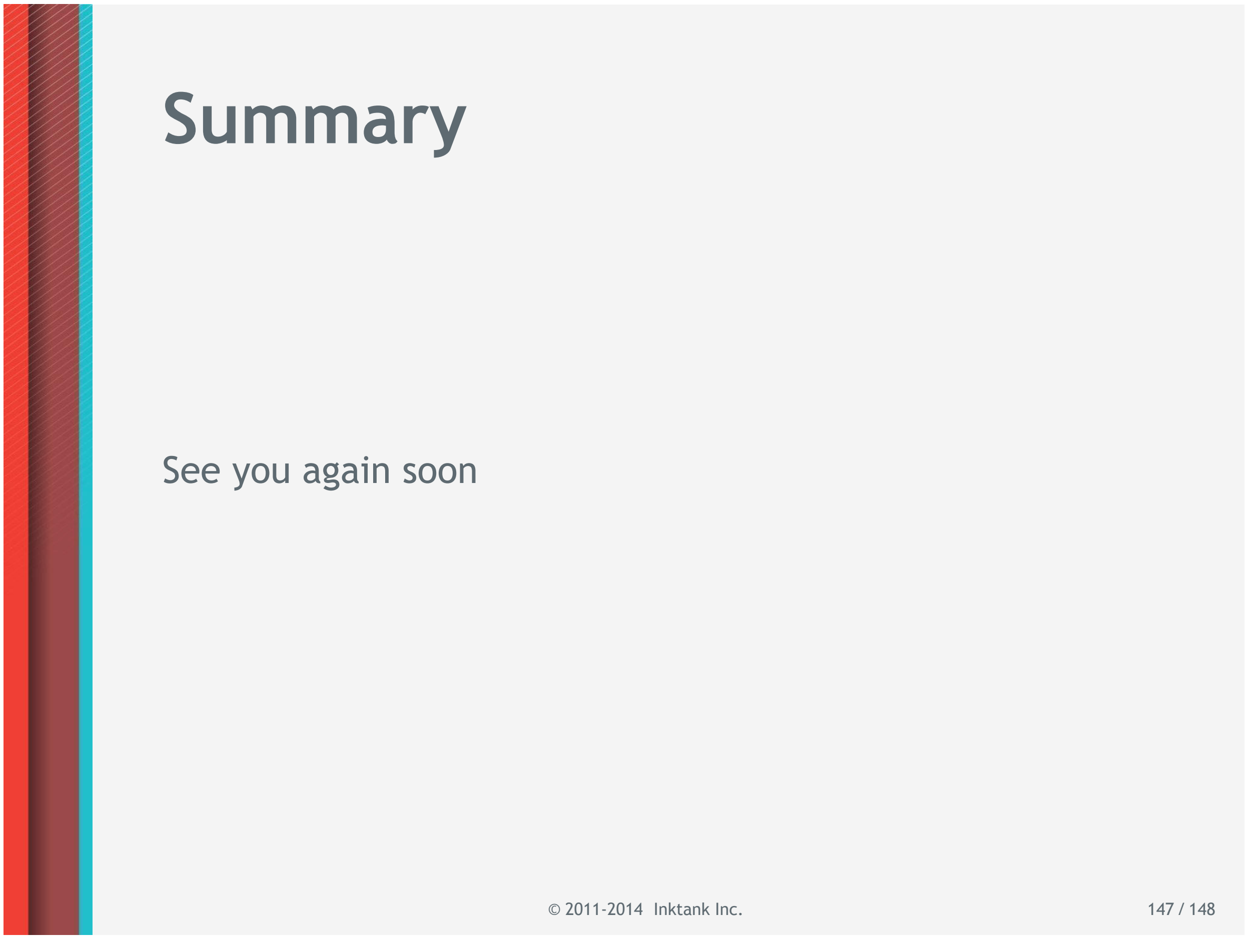
- At the end of this lesson, you will be able to:
  - Tell us how you feel
    - About the slide deck format
    - About the slide deck content
    - About the instructor
    - About the lab format
    - About the lab content



Please Tell Us

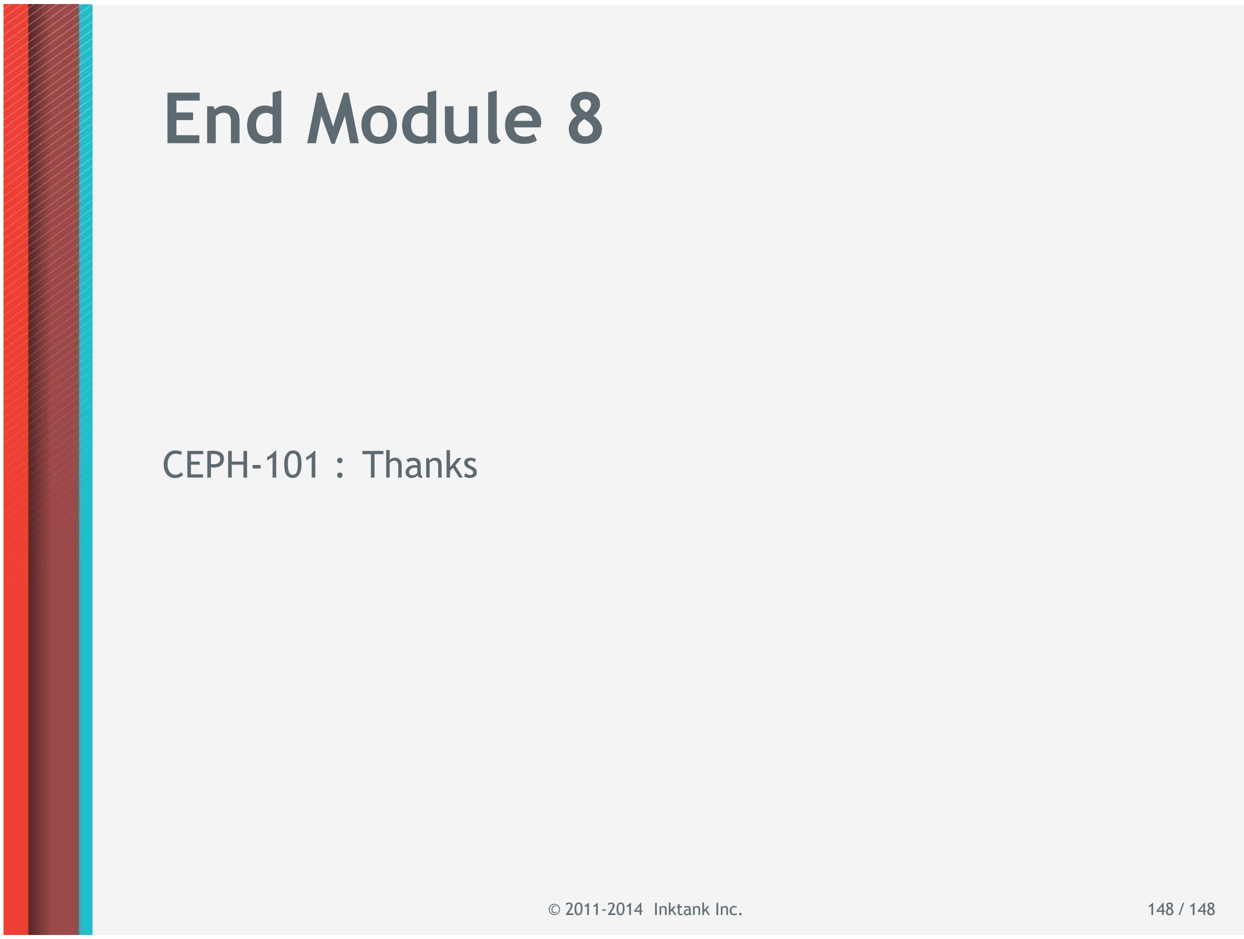
We hope you enjoyed it !

- Tell us about your feedback
  - `http://www.inktank.com/trainingfeedback`
- Tell us about what we could do better
  - Q&A



Summary

See you again soon



End Module 8

CEPH-101 : Thanks