I’m Greg; I’ve been working on Ceph for 3.5 years. I’m going to talk about some of the considerations when designing your cluster. There’s one question that I see a lot in IRC and our mailing lists <click>
What’s the best way to configure a cluster?

My favorite answer is: *shrug*
For a very good reason: <click>
The best configuration depends on cluster requirements.
Considerations

- Required size of storage
- Required storage performance
- Cost considerations
- Failure expectations

People run Ceph on everything from their home computer network (they’re just having fun), to hundreds of servers with petabytes of storage, to a couple racks of mixed storage and compute that are stuff full of SSDs. Each of those deployments is “the best” for them. So here are some things to consider when doing cluster design. (Size, performance, cost, failure)
Components

Of course, a Ceph cluster consists of many different software daemons. You have your monitors keeping track of everybody; your OSDs storing actual data; if you’re running CephFS you have MDSes keeping track of the file hierarchy; and if you’re providing an S3–alike service then you have RADOS Gateways. Each of these needs to be considered separately, and luckily they aren’t really tied to each other much. This is one of the things that keeps costs down compared to a black box storage system. Unfortunately, that does mean you need to know the requirements associated with all of them. We’re going to go through all of these today; let’s start with the monitors.
Of course, a Ceph cluster consists of many different software daemons. You have your monitors keeping track of everybody; your OSDs storing actual data; if you’re running CephFS you have MDSes keeping track of the file hierarchy; and if you’re providing an S3–alike service then you have RADOS Gateways. Each of these needs to be considered separately, and luckily they aren’t really tied to each other much. This is one of the things that keeps costs down compared to a black box storage system. Unfortunately, that does mean you need to know the requirements associated with all of them. We’re going to go through all of these today; let’s start with the monitors.
Of course, a Ceph cluster consists of many different software daemons. You have your monitors keeping track of everybody; your OSDs storing actual data; if you’re running CephFS you have MDSes keeping track of the file hierarchy; and if you’re providing an S3-alike service then you have RADOS Gateways. Each of these needs to be considered separately, and luckily they aren’t really tied to each other much. This is one of the things that keeps costs down compared to a black box storage system. Unfortunately, that does mean you need to know the requirements associated with all of them. We’re going to go through all of these today; let’s start with the monitors.
Of course, a Ceph cluster consists of many different software daemons. You have your monitors keeping track of everybody; your OSDs storing actual data; if you’re running CephFS you have MDSes keeping track of the file hierarchy; and if you’re providing an S3–alike service then you have RADOS Gateways. Each of these needs to be considered separately, and luckily they aren’t really tied to each other much. This is one of the things that keeps costs down compared to a black box storage system. Unfortunately, that does mean you need to know the requirements associated with all of them. We’re going to go through all of these today; let’s start with the monitors.
Of course, a Ceph cluster consists of many different software daemons. You have your monitors keeping track of everybody; your OSDs storing actual data; if you’re running CephFS you have MDSes keeping track of the file hierarchy; and if you’re providing an S3–alike service then you have RADOS Gateways. Each of these needs to be considered separately, and luckily they aren’t really tied to each other much. This is one of the things that keeps costs down compared to a black box storage system. Unfortunately, that does mean you need to know the requirements associated with all of them. We’re going to go through all of these today; let’s start with the monitors.
Components

These all scale independently!
...but they have their own rules for configuring, too.

Of course, a Ceph cluster consists of many different software daemons. You have your monitors keeping track of everybody; your OSDs storing actual data; if you’re running CephFS you have MDSes keeping track of the file hierarchy; and if you’re providing an S3–alike service then you have RADOS Gateways. Each of these needs to be considered separately, and luckily they aren’t really tied to each other much. This is one of the things that keeps costs down compared to a black box storage system. Unfortunately, that does mean you need to know the requirements associated with all of them. We’re going to go through all of these today; let’s start with the monitors.
Let's build up the cluster. First, the monitors, who are responsible for keeping track of cluster membership for everybody else.
Monitor nodes require

- A mounted hard drive.
- Yes, this can be the OS drive.
- A defined IP address.

There are two categories of configuration for each daemon in the cluster. The first is the node configuration — local things like hard drives and network ports. Monitors are pretty lightweight in this respect. They need a filesystem to store their data on; it will see a fair number of fsyncs but the monitor doesn’t need it to be particularly high-performance. You also need a fixed IP address — these IPs uniquely identify the monitors and are how all the other nodes join the cluster.
The second area to configure is the cluster layout of each daemon. I want to do a quick illustration of how monitors do their cluster maintenance so you understand why the cluster requirements work the way they do. Basically, one of them is the leader, he <click>
Monitors

says to the others “write down this state of the cluster”, then they say “aye aye, sir!” <click>
Monitors
do so, and tell him. This is logically a voting process (a very dictatorial one!), and since the monitors were born in the United States the vote only passes if more than half of them agree — a plurality is not sufficient.

This “more than half” rule is what prevents the split brain problems seen by some systems, by the way.
do so, and tell him. This is logically a voting process (a very dictatorial one!), and since the monitors were born in the United States the vote only passes if more than half of them agree — a plurality is not sufficient.

This “more than half” rule is what prevents the split brain problems seen by some systems, by the way.
If one of the monitors in this example goes down, or is somehow network partitioned, then the vote can still pass.
Monitors
Monitors
Monitors

2 of 3: Vote passed!
But if a second one goes down (and notice this is the last scenario from the other perspective), he can’t pass a vote for anything he does.
Monitors
Monitors

- ✔️ Monitor
- ✗ Monitor
- ✗ Monitor
Monitors

1 of 3: Vote failed! :(

M

M

M
This behavior is the most important thing to know when designing your monitor cluster. So you might ask a question: <click> I’m excited by this question, because I have a useful answer when people ask this one: Three! Three! The answer is Three!
How many monitors do I need?

This behavior is the most important thing to know when designing your monitor cluster. So you might ask a question: <click>
I’m excited by this question, because I have a useful answer when people ask this one: Three! Three! The answer is Three!
You might wonder at my enthusiasm, and ask <click> if I’m sure. Pretty sure, because if your cluster was big enough to need more monitors on a size basis you should be talking to us, and you aren’t. ;)

If you’re particularly clever you might have spotted the flaw with this argument. <click>
My cluster’s big, are you sure I don’t need 5?

You might wonder at my enthusiasm, and ask <click> if I’m sure. Pretty sure, because if your cluster was big enough to need more monitors on a size basis you should be talking to us, and you aren’t. ;)

If you’re particularly clever you might have spotted the flaw with this argument. <click>
I don’t trust my servers; don’t I need more than 3 to withstand failures?

You got me. Yes, if you’re really concerned about your monitor servers going down, you should do more than three. Hopefully your monitors are well cared for so that this doesn’t happen though — my sample size is small but I haven’t seen a situation yet where three monitors failed but five would have continued. <click>
Where do I put my monitors?
Don’t do this.
Something like this is a better choice. You want to scatter your monitors through all the different failure domains you have: different switches, different power supplies (and circuits!), whatever you can do to make sure they won’t fail unless your whole cluster is already down anyway. If you have multiple data centers in a single high-speed network, they should be in different data centers.
Basic rules

• Only use an odd number of monitors (>1)
• Use as few monitors as your resiliency requirements allow
• Place your monitors far apart from each other.

You use an odd number because even numbers don’t protect against a larger number of failures, but do bring in more nodes that can fail. You should use fewer monitors just for simplicity, and although it’s not a big deal, to protect against the long-tail problem. Your monitors should be far apart so they can’t get taken out by one electron with a grudge.
So that’s the monitors; let’s turn our attention to OSDs.
OSD Nodes Require

- An OSD Journal (block device or file)
- A backing store (a filesystem)
- Network connectivity
- Some CPU/RAM
What does “Some” CPU/RAM mean?

It means we’re not entirely sure. Our rule of thumb estimate is that you should have 1GHz of CPU and 1GB of RAM per daemon. However, it’s a somewhat generous estimate because we don’t want people to underprovision — Wido here has done a lot of work running 4 daemons against a dual-core Atom CPU, and he’s had problems in the past but is seeing success today. And that’s certainly something we’d like to see succeed.
What does "network connectivity" mean?

- TCP/IP connectivity to the cluster
- Separate OSD and "everybody else" traffic if you like
- Speed appropriate for your node's disks and cluster speed requirements.

You need an ethernet connection, basically. If you have Infiniband, you’ll need to set up IPoIB. The OSDs do segregate their inter-OSD communication from their client and monitor communication, so though it’s not required you can set up different physical networks for that.

The speed depends on your requirements. Most of our customers are using 10Gb, but if you have small nodes (we’ll get to that) or you don’t mind a slow pipe (maybe you’re building a Backblaze competitor) you can run on 1Gb.
What does "a backing store" mean?

- A filesystem where all OSD data is stored
- xfs, btrfs, ext4

[Talk about performance degradation with btrfs, bad xattrs (but maybe no longer a problem) on ext4, and that current production installs are on xfs. Mention zfs?]
So I should RAID all my drives together?
Probably not. Instead, we normally recommend running one OSD per disk in your storage node. There are several reasons for that. <click>
• Who wants to RAID and then replicate?

• Random IO on RAID5 is slow...
  
  • ...and if you're mirroring, why aren't you doing it through RADOS?

• Resilvering a RAID volume takes a long time and degrades performance compared to distributed re-replication.

First, we do replication inside of Ceph and it’s not clear there’s a benefit to doing RAID underneath that. You’re usually better off from a data loss perspective with replication rather than RAID.

If you lose a disk in RAID, it’s true that the recovery is entirely local. But you need to go out and plug in a new disk before it can even start, and then you have to wait for a whole drive’s worth of data to write out, which can be a day or more. During this time, you’ve degraded performance for all data accesses to the node, and that’s just annoying. With Ceph, you use the network for replication but it means you can fully re-replicate your data without plugging in new disks, and it completes faster because there are more disks participating in recovery. Finally, RAID5/6 random IO can be pretty slow — and if you’re just mirroring, you should do that with RADOS’ replication.
When you might RAID

• You have too many disks to run a daemon for each.
• You have ultra-reliable storage and aren't replicating across OSDs.
• You want to be different from everybody else.

All that said, there are some reasons to consider using RAID in a Ceph cluster. The first and easiest is if your node has too many disks to run a daemon for each. Remember, we recommend 1GHz of CPU and 1GB of RAM for each daemon; if you have a big 64-drive box you might not be able to satisfy those numbers, so you might want to do a number of smaller RAID arrays.

Second, some people are looking into using Ceph for Hadoop processing or other “scratch spaces”, or even just using ultra-reliable hardware, so they don’t want to pay the cost of any replication at all. In that case, you probably still want some amount of redundancy, so you could RAID your disks.

Third, I say “be different” to be funny, but it actually is useful. Perhaps you want more reliability than 2x replication but 3x replication is too expensive to store. Nobody in the world has experience yet running Ceph with RAID as an important part of the stack, but it could turn out to be the right formula for some workloads. But we know it’s not necessary, so you shouldn’t do so without thinking through what it means and why you want it.
What's the OSD journal look like?

• For consistency, an OSD writes everything to a journal as well as to the backing store.

• If your journal is slower than your backing store, it will be a limiting factor.

• But you can make it a file or use a raw block device
So can I use a RAM disk?

Don’t do that. <click> The OSDs consider anything written in the journal to be durable, and make decisions based on that. If you lose the journal with xfs or ext4, you’ve lost the OSD because we can’t guarantee that the node is in any kind of consistent state — it expects to be able to replay operations. With btrfs, it’s not quite as bad — you’ll go back in time. That’s nice if you just lose one node; most of the data will be there and it can quickly recover from its peers on restart. But consider if your data center power flaps and you lose all your nodes at once — is that data loss acceptable?
So if you can’t use a RAM disk, let’s talk about what you can do.
So can I use a RAM disk?

- If you lose a journal with an xfs or ext4 backing store, you lose that OSD.
- If you lose a journal with a btrfs backing store, it goes back in time.

Don’t do that. <click> The OSDs consider anything written in the journal to be durable, and make decisions based on that. If you lose the journal with xfs or ext4, you’ve lost the OSD because we can’t guarantee that the node is in any kind of consistent state — it expects to be able to replay operations. With btrfs, it’s not quite as bad — you’ll go back in time. That’s nice if you just lose one node; most of the data will be there and it can quickly recover from its peers on restart. But consider if your data center power flaps and you lose all your nodes at once — is that data loss acceptable? So if you can’t use a RAM disk, let’s talk about what you can do.
Wait, you haven't told me what kind of disks to use!

- Match them to your performance needs.
- The journal can disguise latency, but not add sustained IOPS.
- Common options:
  - SSD (often split among multiple daemons)
  - Separate partition or RAID group on data store disk
  - File on data store filesystem

Basically, you should pick the storage that matches your performance needs. If you’re hosting virtual images, you probably want your journals to be a bit faster than your backing drive so that you can absorb spikes in workload quickly and efficiently.

That leaves a lot of room for deciding what kind of configuration you want to set up. SSDs split up between 3–5 drives are a popular option because they handle small writes so much better than HDs — Wido’s Atom cluster uses an SSD and 4 large disks. DHO splits each of their data disks into a small journal partition and a large storage partition, which doesn’t perform as well but localizes risks of failure and minimizes costs. You can also just use a file located somewhere, although we recommend going with as much separation as you can — a separate partition can be given different writeback settings with a RAID card, for instance, and is likely to handle random writes better than a file that has to go through the FS.

And if you’re spendy, one of the original plans for a journal was just to use NVRAM. <click>
How big?

Your journal is a buffer. It needs to last through backing store syncs — running every five seconds by default — and other interruptions. 10GB is a common size and up to a point bigger is definitely better, but you should be good if you have space for 10–30 seconds of writes. In particular this means that most SSDs can be under-partitioned, which prolongs longevity quite a bit.
Let's talk CRUSH!

It’s awesome!
• Model your failure domains:
  • default list: OSD, host, rack, row, room, data center, “root”

• If you want, set OSD tiers

With CRUSH, you want to model your failure domains. We have a bunch of default “buckets” to group together nodes in different domains, but you don’t need to use all of them. Most deployments today stick with OSDs, in hosts, in racks. Then you can set up your CRUSH rules to separate OSDs across whatever level you like. More advanced options are also available; for instance you can have a rule that takes one replica out of your “SSD” bucket and takes the rest of the replicas from different racks in the “standard” bucket.
Let’s talk overall node configuration!

You want to deploy the cheapest configuration that suits your storage needs. I can’t possibly tell you what that is right now, but you probably want to look at things from a rack level rather than a node level.

If I were setting up a system, I would choose the one with the highest disk density that approximates those CPU and RAM requirements we discussed before.
If you’re experimenting with the filesystem, you need to set up MDS servers too.
MDS Nodes Require

- Network connectivity
- A lot of RAM
- CPU power

MDS node requirements are a lot simpler than OSDs. The more RAM you give them, the more of the filesystem hierarchy they can cache. The faster the CPU, the more ops/second they can handle. The MDS doesn’t move much data around so it doesn’t need a super-fast network connection under most use cases.
How many MDSes?

One. Ceph’s MDS system is a cluster design and is meant to scale, but it has a lot of QA to go through so a multi-MDS cluster is many times more likely to break than a single-MDS system is. <click>
When it’s stable?

• Scale your MDS cluster based on active usage — not total data or tree size!

People sometimes come in to irc and say “I have 300 TB of storage, how many MDSes do I need?” That’s not how it works. The MDS is a cache and all data is stored in RADOS, which means your MDS cluster needs to be large enough for two things:
1) to hold your entire working set in memory,
2) to provide the number of metadata operations your cluster is pushing.
We haven’t run recent experiments, but based on Sage’s thesis work an MDS can push a few thousand ops/second.
I saw something about “standby”

Remember, the MDS has no local data, so if a daemon dies you can start it up again on any system that has cluster access. We provide options to make this super easy — if you run more MDS daemons than are configured to be “active”, the rest will go into “standby” and take over if another node fails. For even faster recovery you can specify that a standby node should follow one of your active nodes, which will keep the same data in memory to pre-populate the cache.
The last daemon in our tour is the RADOS Gateway.
The RADOS Gateway is a proxy which turns HTTP requests into RADOS requests and vice-versa. They can use memory to form an effective cache, but they use that for metadata so you don’t need a ton — you should use a separate cache in front for full objects (and support for this is coming). Deploy as many as you need and stick them behind a load balancer — they are fully cluster-aware.
Deployment Options

• mkcephfs
• ceph-deploy
• Chef cookbooks
  • Crowbar
• Juju charms
• manually (presumably a different system)
mkcephfs is the original utility that we wrote to deploy test ceph clusters. It uses old standards of deployment, deciding where everything in the cluster goes before you turn it on and then placing the same config file on every node. This does give it some advantages in terms of setting the right configuration options automatically, but it means that mkcephfs can’t handle things like adding three nodes to your cluster — it’s just for an initial deployment.
Our new shiny
(with all that entails)
Chef

- Infrastructure & Configuration as Code
- DreamHost uses Chef recipes

Chef is one of the big configuration management systems today. We have cookbooks available on our github account to manage this and they are good Chef citizens, correctly handling adding new nodes to the system and configuring the daemon locations correctly. Chef is how DreamHost is managing their DreamObjects deployment of ~800 daemons and I believe they’ve also open-sourced their own recipes.
Crowbar

Crowbar is a tool from Dell that layers on top of Chef to add management features. We’ve written “barclamps” for it and they work pretty nicely as well. These are available on Github (check out my page as well as the Ceph account) and should be made available to Dell customers in their next release.

SUSECloud is also built on Crowbar and the kernel RBD implementation, and you can see those barclamps on their GitHub page.
Juju

• “Service Orchestration”
• Canonical wrote some Charms
  • They’re available in the Charms Store

Juju is an exciting “service orchestration” tool from Canonical which focuses less on configuring a node the right way and more on configuring your services. I’m very excited about it just from a development point of view, and Canonical has already produced charms which we’re going to be jointly maintaining in the future. They make it very easy to test Ceph in virtualized environments, and to deploy it on hardware with their “Metal as a Service”.
Manual deployment

So you probably don’t really want to build your system manually, but if you know how to do it manually you can write your own scripts for Puppet or Fabric or something you’ve built internally.

This actually turns out to be not terribly difficult; I’ve walked people through deploying a whole system in a single afternoon. You don’t even need a config file; you just need a monitor address and any keys that are required for the daemons.
Questions?

I’ll take questions now, but first a reminder that the next session after a quick coffee break is going to be a panel on deployment so you might want to save some questions for that to get a different perspective.