Availability Goals

Alarms

Summary: 58 total alarms for LDAP 1/2, CPR 1/2, IdP 1/2, and CPR's two databases.

WARN (to ndk@internet2.edu and i2commit-warn@unicon.net):

CPU Util >= 50% mean checked every 5 minutes
Disk Util >= 80% checked every 5 minutes
RAM Util >= 85% checked every 5 minutes
Swap Util > 0% checked every 5 minutes

CRIT (to ndk@internet2.edu and i2commit-crit@unicon.net):

StatusCheckFailed >= 1 checked every 5 minutes
CPU Util >= 80% mean checked every 5 minutes
Disk Util >= 90% checked every 5 minutes
RAM Util >= 95% checked every 5 minutes
Swap Util > 5% checked every 5 minutes

Availability approach:

The goal is to ensure users can always log in. This means a) clustering everything possible, b) maximizing individual server uptime, c) scaling to meet load as required and d) anticipating catastrophic failure scenarios and having a rapid recovery strategy.

a) Clustering everything possible means… we cluster everything possible.

The load balancer (ELB) has built-in clustering. When the need is felt to move to ha proxy, that means running multiple ha proxy servers with round-robin DNS in front of them.

The IdP is a stateless cluster and each user is associated with a single IdP. Failures of an IdP will result in that IdP being failed out of the pool(which can happen quickly), along with the loss of any active user sessions. For users who were already logged in, this will just mean they lose their SSO and will need to re-authenticate. In order to improve on this, we would need to implement a login-level session storage mechanism like the OSU stateless login handler. The worse problem is that, for users in the process of logging in, they will get dropped. The only fix for that is full session replication between IdP's, which introduces significant load and its own complications. This may be a requirement beyond the pilot.

Jetty is not clustered and doesn't need to be.

389 has multi-master replication, so I don't believe a server failure will result in a noticeable outage.

If we loose the CPR SQL database it will fail over to the other node, but I think we will need to do something in the configuration for it to use it. As for ActiveMQ, that will just happen automatically with no changes. We have tested that at Penn State with no problems. So right now the database is the only thing we would need to worry about. I have access to a PostgreSQL DBA back at work that I can ask about this.

These failure modes extend to deliberate service restarts in addition to unplanned outages.

b) Maximizing server uptime will principally be a matter of performing extensive load testing, particularly for long durations of time, to see if any issues arise.

Incremental updates should be used so that in the event there's e.g. unloadable configuration or a bug in new software, only one node is lost and service is maintained.

c) We will need a mechanism that senses how much load a component is under and is capable of either spinning up additional nodes of it(our preference, but maybe hard to do), or notifies an administrator to add more nodes.

It's my belief that when a server fails in a hosted environment like EC2, we don't need to treat it any differently than load increasing on the rest of the cluster (all the users go there when their existing server croaks). This is something we are building for anyway. I'd just let the dead server lay there for later autopsy.

I think Michael is building this for the IdP, Jetty, and 389. I don't know who's doing it for the CPR pieces.

d) Handling catastrophic failure modes is probably the most difficult. We can and will test various failure modes, but it's hard to ancitipate all the things that could go wrong.

It's better to have the ability to quickly respond to catastrophic failures. Some sites will maintain hot swappable servers. We can make a decision whether the incremental cost for this is justified by the added availability – or whether it really adds any availability at all versus just having these nodes be part of the cluster anyway.

For all seriously horrible events, human intervention will likely be necessary. I don't know how or who but we should plan for this.

Logging:

The general goal will be to log as much centrally as possible while maintaining distributed logs, trying to maximize benefits of each approach

In practice with the IdP and jetty, this will be adding a remote syslog appender that logs to a centralized host. This will be done in concert with local logging with an aggressive rotation schedule. In the event that the centralized host is unreachable for some reason, the processes will be fine and continue to log locally.

389 doesn't have this capability, but we can use a pipe to send logs to syslog and replicate the functionality that is built-in to the IdP and jetty.

Right now the CPR uses log4j, so we can hook anything back up against it. At Penn State for our production services like Kerberos and email, they tried a Syslog over TCP cluster using HSM and from what I was told it would not keep up with the millions of authentications that happen daily so they switched back to UDP.

From a logging perspective for the CPR, we can pretty much do anything, right now it logs at service call (which we are not doing), APIs (which we are doing), and parts of the UI to local files. For the production CPR, that will be in place on 1 June, they are doing a syslog cluster, but I am not sure what kind yet.

We need to investigate failover modes here. Syslog over UDP doesn't know whether a write is successful and thus can't failover to a backup server. It may be possible to do syslog over TCP but I don't know whether this is a desirable choice since it increases overhead. We need to do more research and I need to think about it.

Worst case scenario is we lose some logs if we don't get a centralized server working quickly enough and the rollover on clients happens too quickly. This will rarely if ever occur.