December 09, 2014

DRAFT

Mistify Management Architecture

OmniTI

TABLE OF CONTENTS

API SERVICES

There are 3 API Services that are included in and governed by the management architecture:

- **End User (EU) API** - This is the API that is used for create/delete/reboot/etc vm's.
- **Operator Admin (OA) API** - For the cloud "operators" to use for things like adding hypervisors, removing users, etc. (Some of this could actually be text config files.)
- **Internal Management (IM) API** - Hypervisors, etc, hit this rather than hitting the database directly. Generally for hypervisor related tasks

Each of the API's will be a separate service. This allows for a clean separation of code, separate security models, etc. All three will use Postgres as their main data store.

SUPPORTING LIBRARIES

- **Metrics** - Every service and every virtual machine (VM) is (or will be) instrumented. For now, we have decided just to use local Statsd instances. We will want to develop a generic library for services so that code in the service doesn't need to change if we change topology, etc.

- **Discovery** - We need a library (and possibly a service) for each component to find ("discover") the other components. We've discussed using SRV records, consul, etcd, etc. We may just do a stub library for now. The components should be built with the concept of being discoverable and needing to discover other components in mind. This heightens the need for us to get something pretty quickly implemented before continuing to far down the road in development.

- **Postgres Replication / Failover** - Initial though here is to port Joyent's open source Manatee to Go combined with consul/etcd. Looking for feedback on this concept. Recommended name for this would be Mirounga - http://en.wikipedia.org/wiki/Elephant_seal.

COMPONENTS

- **Image Service** - Simple front end for Riak-CS?

- **DHCP for Hypervisors -** Just generate configs from the Internal API? No need for a custom DHCP server. Will need at least one per layer 2 network or do relaying.

- **TFTP -** This will be for initial pxe boot. Do we just distribute bits with it?

- **OS Versions/Bits for Hypervisors -** We will set in the Operator Admin (OA) API the version we want each hypervisor to run, we'll need a service for ipxe to call to know which version of the bits to grab. This could be built into the Internal Management API (IM) and store the bits in Riak-CS

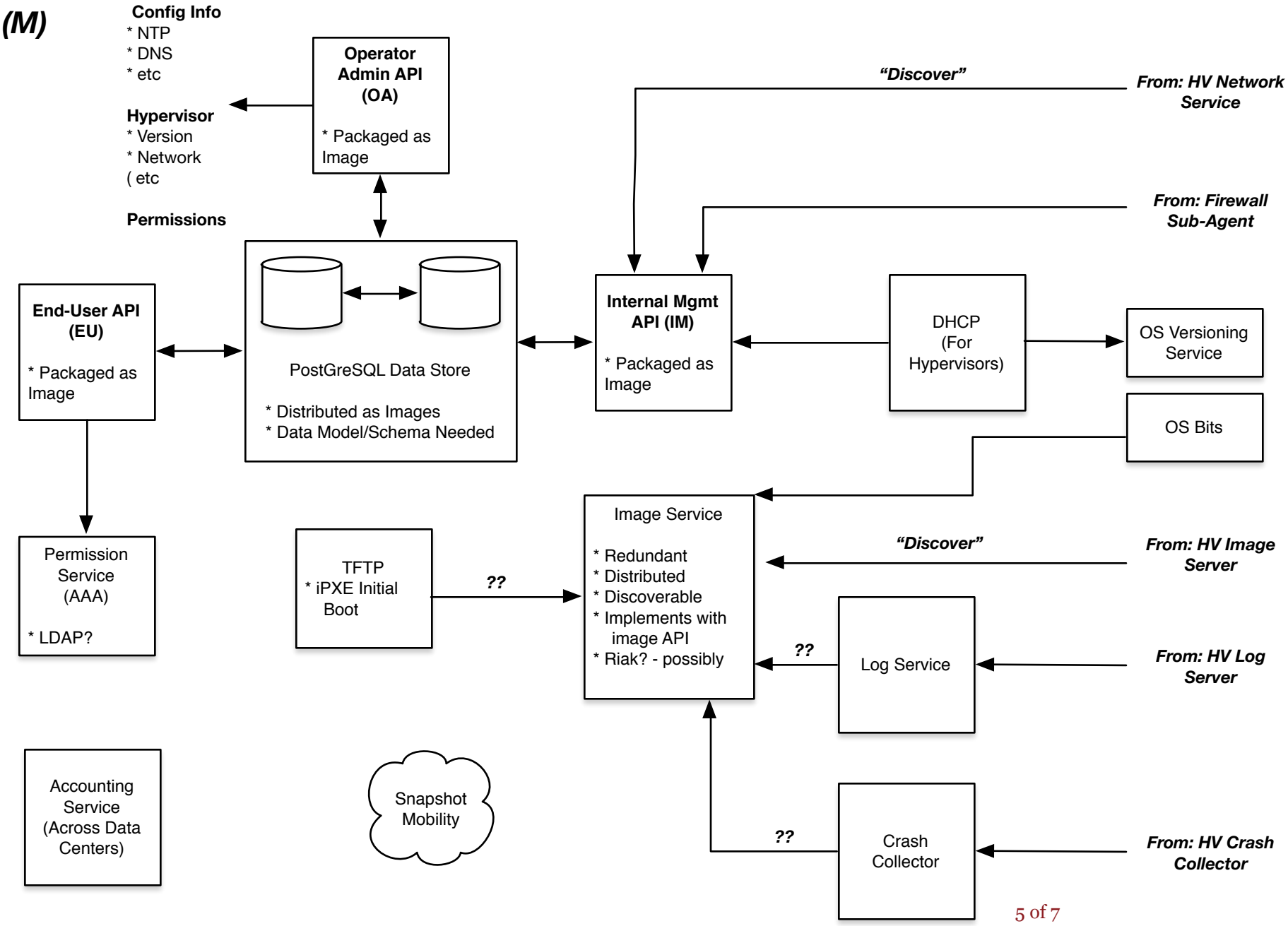
- **Management "Agent Helper"** - Connects current Agent within each HV with management APIs. This could live on each hypervisor or be a separate process, or be built into one of the other API's. I prefer to have each "helper" process only concerned about a single entity, so it is planned that it will run on each hypervisor. I'll expound on this in another email/doc/drunken ramble.

- **Queue (Simple Queuing)** - This really is just to tell other components to wake up and check an API for some information. For example, we may create a "job" and drop it's ID onto a queue and a hypervisor "helper" would pick up this ID, then get the full job info from the IM. We had actually thought about using Postgres notify/listen for this, though it may not be feasible for a large number of nodes.

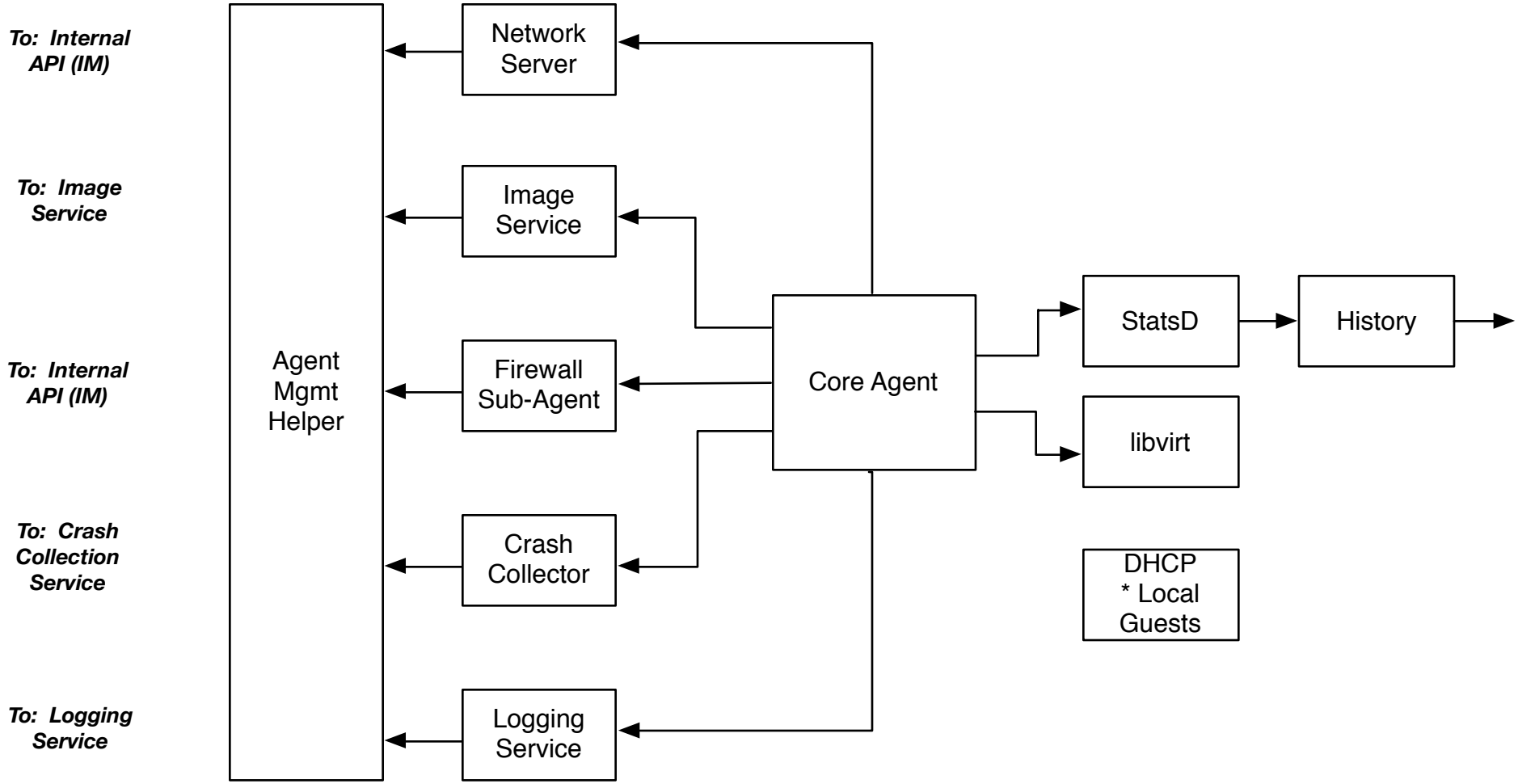
- **Firewall Sub-Agent** - The purpose of the firewall sub-agent is to query the internal management API for firewall rules and update local OS specific access control lists for guests.

- **Other** - For some components, we may want to have leader-election, or an easy way for an operator to say "make sure I have x number of these." Also, perhaps use chef (or similar) to handle config file generation and using the Internal API to drive it.

Management API (M)



Hypervisor


To: Internal API (IM)
To: Image Service
To: Internal API (IM)
To: Crash Collection Service
To: Logging Service
Agent Mgmt Helper
Network Server
Image Service
Firewall Sub-Agent
Crash Collector
Logging Service
Core Agent
StatsD
History
libvirt
DHCP * Local Guests

HV Boot Process