Xen Summit
18 April, 2007
TJ Watson Research Center

The Xen-API

Ewan Mellor
ewan@xensource.com

Xen Management Architecture

Cluster-wide Integration

The aim is to integrate Xen-based systems with enterprise-level cluster management solutions.

These solutions will use CIM as the integration interface. The Xen-CIM effort is of paramount importance.

Xen-CIM needs a stable interface into Xend.

Xen Management Architecture

Lightweight Management

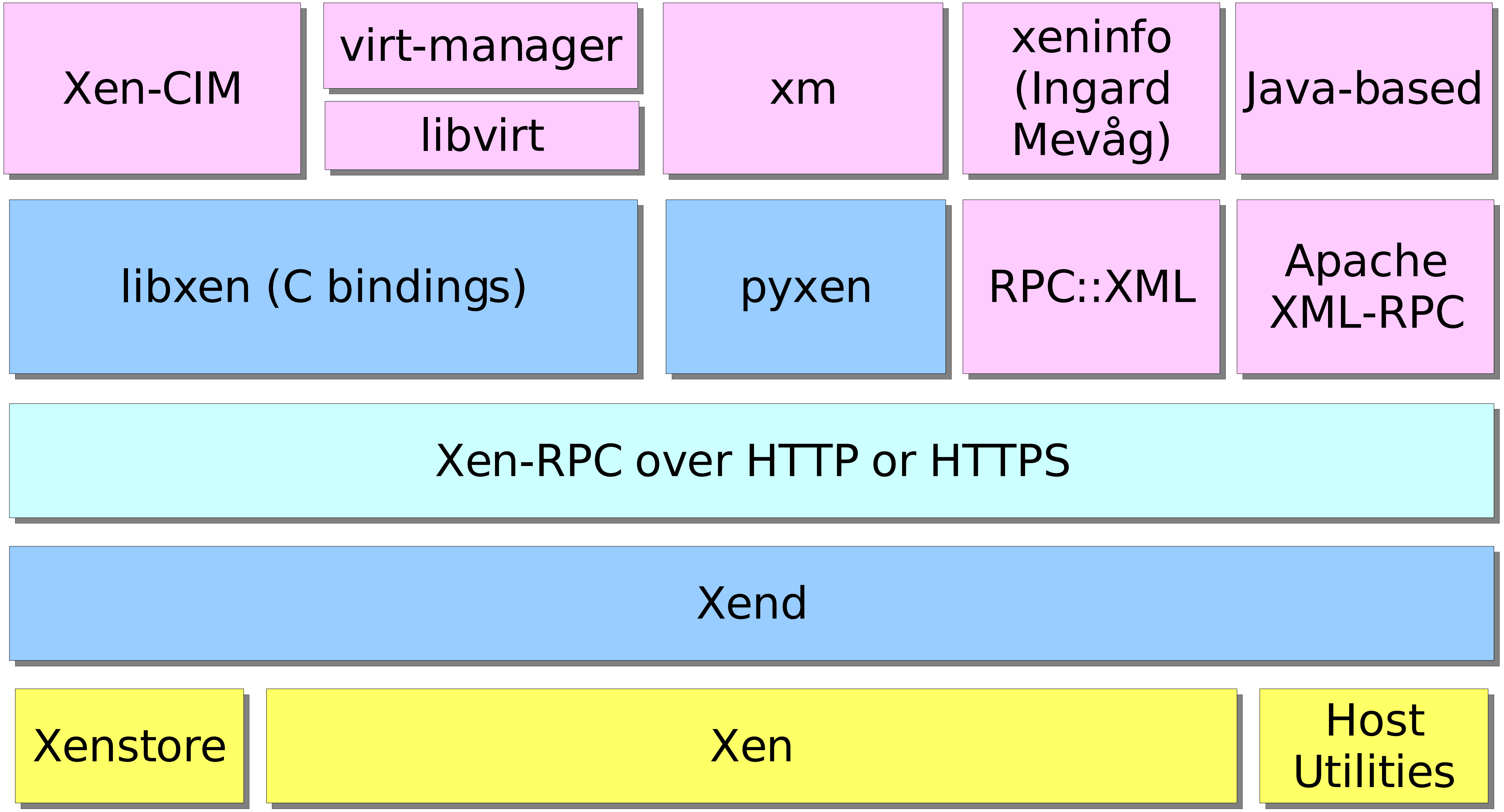
We further aim to make it possible to manage Xen-based systems remotely, without enterprise-level solutions.

Build an ecosystem of third-party tools around Xen.

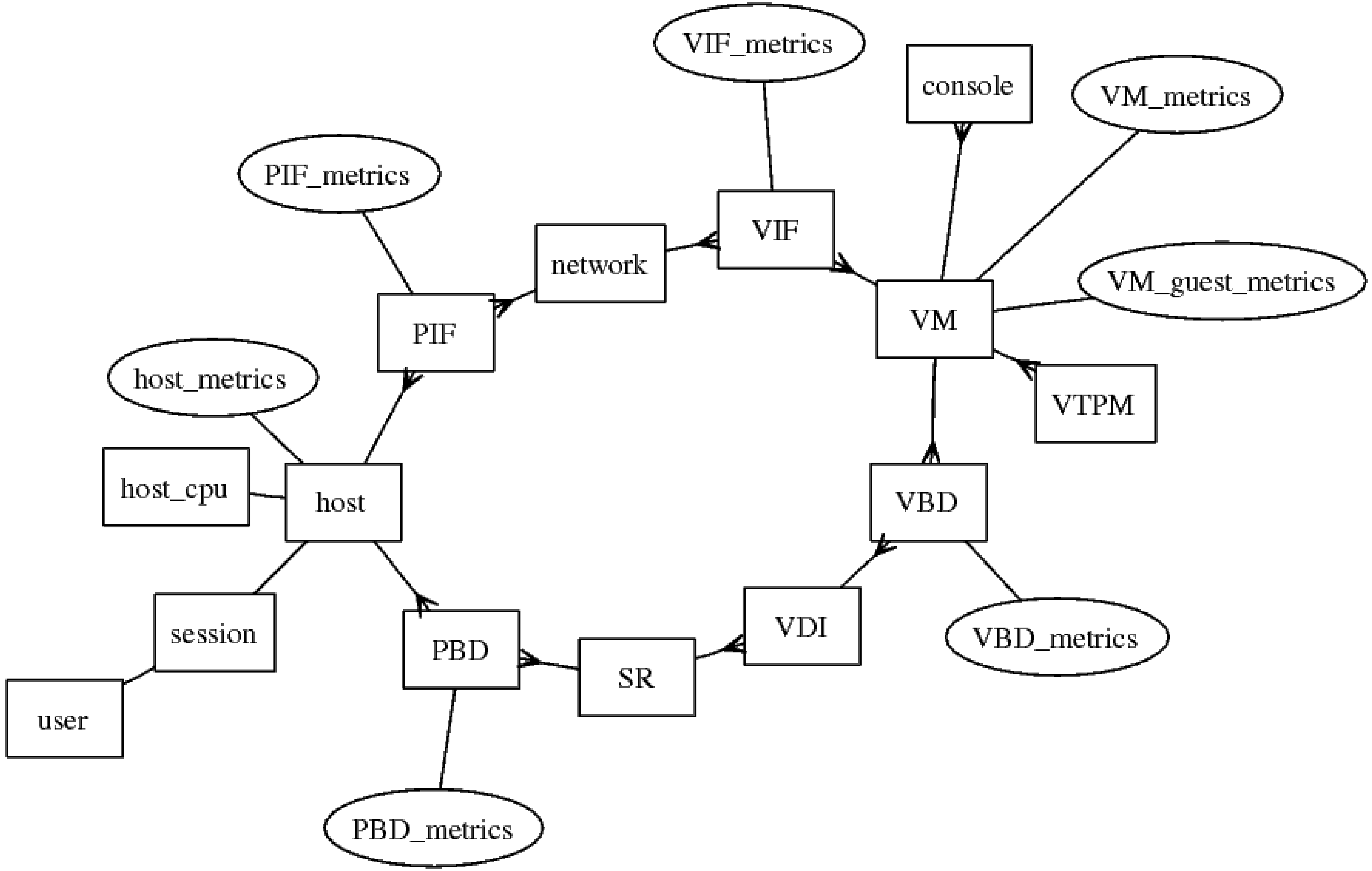
Keep the interfaces stable over the long term, allowing these tools to mature.

Red Hat's virt-manager is the first major application in a position to use the Xen-API, as a libvirt backend.

Xen Management Architecture



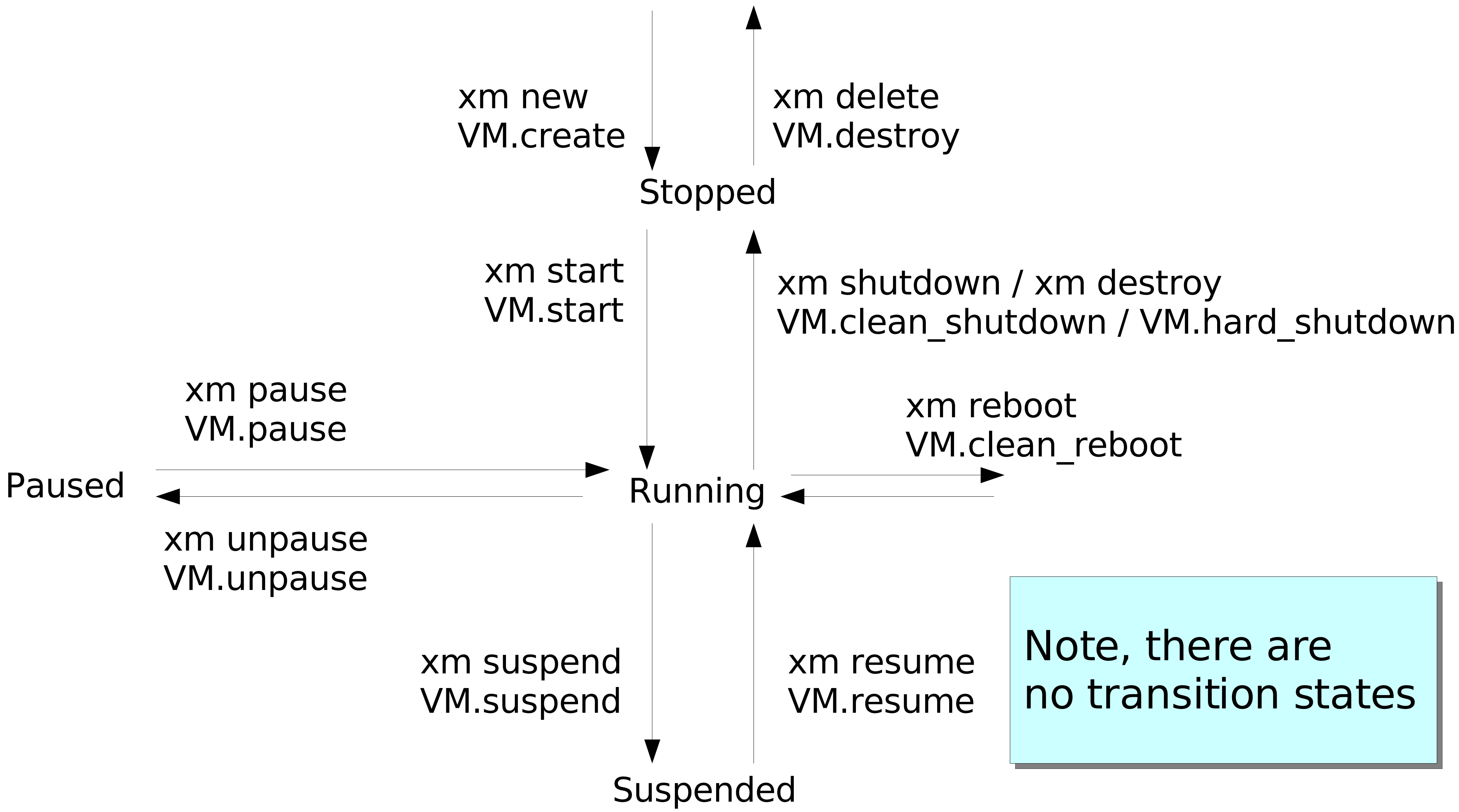
Xen-API Class Hierarchy



Xen-API Basic Example

```
session_ref = session.login_with_password('ewan', 'hello')

vm_list = VM.get_all(session_ref)

for vm_ref in vm_list:
    vm_record = VM.get_record(session_ref, vm_ref)
    print “VM name:” + vm_record.name_label

    VM.start(session_ref, vm_ref, false)

    vbd_list = VM.get_VBDs(session_ref, vm_ref)
    for vbd_ref in vbd_list:
        ...
```

VM Lifecycle



Note, there are
no transition states

Retrieving Metrics

```
~ # xm shell
The Xen Master. Type "help" for a list of functions.
xm> VM_metrics.get_record 53b59b27-adeb-855e-8ea3-a50d4e3b9824
{'VCPUs_CPU': {'1': '1', '0': '0', '3': '1', '2': '0'},
 'VCPUs_flags': {'1': [], '0': ['online'], '3': [], '2': []},
 'VCPUs_number': '1',
 'VCPUs_params': {'cap': '0',
                  'cpumap0': '0, ...',
                  'cpumap1': '0, ...',
                  'cpumap2': '0, ...',
                  'cpumap3': '0, ...',
                  'weight': '256'},
 'VCPUs_utilisation': {'1': 0.0, '0': 0.124, '3': 0.0, '2': 0.0},
 'last_updated': <DateTime u'20070418T13:02:39' at 2aaee5c37ea8>,
 'memory_actual': '268304384',
 'start_time': <DateTime u'20070418T12:59:49' at 2aaee5c37b90>,
 'state': [],
 'uuid': '53b59b27-adeb-855e-8ea3-a50d4e3b9824'}
```

VIF, VBD, PIF also have metrics
Bandwidth values are averages over the last second

Error Handling

All calls return an XML-RPC structure:

Status: “Success”
Value: <the result>

or

Status: “Failure”
ErrorDescription: [“VM_BAD_POWER_STATE”, “Halted”, “Running”]

The first element of the list is the error code.
Each error code has parameters specific to that code.
Error messages can then be internationalised on the client-side.

pyxen captures the Status: Failure and creates an exception.
libxen saves them on the session object.

Asynchronous Callbacks

- “Blocking poll” for receiving callbacks – event.next()
- The client uses their existing HTTP client library
- No need to open a socket in the client

Simple event model:

- This field on this object has changed
- An object was created
- An object was destroyed

Registration on a per-class granularity

Asynchronous Callbacks

xm event-monitor:

```
server.xenapi.event.register([])
while True:
    events = server.xenapi.event.next()
    for e in events:
        pprint.pprint(e)
```

```
{'class': 'VM',
 'field': 'power_state',
 'id': '3',
 'obj_uuid': '8ef25b6f-889f-a1e0-6312-4bb532ed7d3f',
 'operation': 'mod',
 'ref': '8ef25b6f-889f-a1e0-6312-4bb532ed7d3f',
 'timestamp': <DateTime u'20070418T12:59:49' at 2aef2111a3b0>}
```

Configuration

Xen-API XML-RPC over HTTP / TCP on port 9363, using PAM and an allowed-hosts specifier

In /etc/xen/xend-config.sxp:

```
(xen-api-server ((9363 pam '^localhost$ example\\.com$')
                 (9367 pam '' /etc/xen/xen-api.key
                              /etc/xen/xen-api.crt)
                 (unix none)))
```

HTTP over TLS over TCP on port 9367, using PAM and the specified keys

Xen-API XML-RPC over a Unix domain socket at /var/run/xend/xen-api.sock, unauthenticated

Example in C

```
xmlInitParser();
xen_init();
curl_global_init(CURL_GLOBAL_ALL);

xen_session *session =
    xen_session_login_with_password(call_func, NULL, username, password);



xen_vm vm;
if (!xen_vm_get_by_uuid(session, &vm, uuid)) {
    /* Error */
}

xen_vm_record *vm_record;
if (!xen_vm_get_record(session, &vm_record, vm)) {
    /* Error */
}

if (!xen_vm_start(session, vm)) {
    /* Error */
}
```

On-wire Example

```
xen_vm_record *vm_record;
if (!xen_vm_get_record(session, &vm_record, vm)) {
```

```
<?xml version='1.0'?>
<methodCall><methodName>Vm.get_record</methodName><params>
  <param><value><string>12345678-abcd-ef90-123456789abc</string></value></param>
</params></methodCall>

<?xml version='1.0'?>
<methodResponse><params>
  <param><value><struct>
    <member><name>Status</name><value><string>Success</string></value></member>
    <member><name>Value</name>
            <value><struct>
              <member><name>name_label</name>
                      <value><string>My VM</string></value></member>
              <member><name>boot_method</name>
                      <value><string>kernel_external</string></value></member>
```

Perl Example

From Ingard Mevåg's xeninfo script.

```
require RPC::XML;
require RPC::XML::Client;

sub get_host_mem_utilisation
{
        my ($xen, $session, $host_name, $host_ref) = @_;
        my $host_metrics_ref =
            validate_response($xen->simple_request("host.get_metrics", $session,
                                          $host_ref));
        my $host_mem_total =
            validate_response($xen->simple_request(
                                            "host_metrics.get_memory_total",

                                            $session, $host_metrics_ref)) / 1024 / 1024;
        my $host_mem_free =
            validate_response($xen->simple_request(
                                          "host_metrics.get_memory_free", $session,
                                          $host_metrics_ref)) / 1024 / 1024;
        $host_info{$host_name}{'memory'} =
            {'total' => $host_mem_total, 'free' => $host_mem_free};
        print "Total: $host_mem_total MB – Free: $host_mem_free MB\n";
}
```

Java Example

Using Apache's XML-RPC library.

```
Object [] params;
HashMap<String, Object> result;

URL url = new URL("http://my.machine:9363");
XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
config.setServerURL(url);
XmlRpcClient client = new XmlRpcClient();

params = new Object [] {"ewan", "hello"};
result =
        (HashMap)client.execute("session.login_with_password", params);
String session_ref = (String)result.get("Value");

params = new Object [] {session_ref};
result = (HashMap)client.execute("VM.get_all", params);
Object [] arr = (Object [])result.get("Value");
int i;
for (i = 0; i < arr.length; i++) {
        System.out.println("VM ref: " + (String)arr[i]);
}
```

Adapted from Stefan Berger's recent xen-api post.

Future Work

Release Xen-API 1.0, with Xen 3.0.5.

Integrate XSM/ACM and XSM/Flask.

Add raw counters to our metrics classes.

Add support for grabbing crash dumps.

Add bindings for other languages. (What do people want?)

More examples and documentation.

Mailing list: xen-api@lists.xensource.com
Wiki: http://wiki.xensource.com/xenwiki/XenApi
Documentation: xen-unstable.hg/docs/xen-api
C bindings: xen-unstable.hg/tools/libxen
Python bindings: xen-unstable.hg/tools/python/xen/xm/XenAPI.py