Infrastructure Independent
Application life cycle

bennys@ravello-systems.com
agenda

• Overview
• Product
• Technology
• Use cases
• Questions
background

ravello systems

- Founded in 2011
- Corporate HQ - Raa’nana, Israel US HQ – Palo Alto, CA
- Deep expertise in virtualization, networking and storage technologies
VHCS – Virtual Hybrid Cloud Services

• **Requirement:** Infrastructure independent application life cycle
  – Any application, Any Stage, Any Cloud, Any Management

• **Challenges:** Heterogeneous Cloud environments
  – Various private environments
  – Multi public cloud platforms (IaaS), oversubscribed with poor SLA
  – Multi public cloud vendors and architectures
  – Maintain existing IT best practices
VHCS major requirements

• Mobility
  – The ability to seamlessly and instantaneously move environments

• Formal requirements and constrains driven service:
  – SLA
  – Security, governance, policies

• Centralized and unified Billing

• The ability to act on cost changes

• Self provisioning

• Management
VHCS existing solutions

1. Massive porting projects
2. Rebuild the app per target environment (CM)
3. Application re-write
4. PaaS
5. Ravello
Capsule
definition: cloud application

web tier

application tier

db tier

storage

router

firewall

load balancer

web

OS

web

OS

web

OS

web

OS

app

OS

app

OS

app

OS

db

OS

db

OS

storage
Any cloud – Amazon (Xen), Rackspace (Xen), HP Cloud (KVM), Google (KVM), VCd (ESX)...
Connect to other services – like before
The power of the capsule

• Infrastructure independent application life cycles.
  – Develop once, run anywhere
• Easily create copies of your app
• Maintain same IT operations and methodologies
  – Easy integration with existing tools
• Blueprint, snapshot full multi-VM application instances.
• Provides the best fit between app, constrains, policies and underlying clouds
• Burst your app to any cloud environment
service based offering
here is how we do it
1. HVX – high performance, nested hypervisor

- Requires no changes to the application or Operating System
- Run any VM (e.g. VMware) in any cloud (e.g. AWS/Xen)
- Supports VM consolidation
2. IO overlay

- Full overlay network (SDN)
  - Mesh of interconnected HVXs
  - Define any L2 and L3 network configuration without dependency on the underlying cloud
  - Supports multicast, broadcast etc.

- SDS - Supports storage abstraction and cloud specific optimization
Dynamic binary translation

• Translates guest binary code into platform compatible code
  – 32bit guest into 64 bit host
  – Replaces privileged instructions
  – CPU registers are shadowed
• HVX reads and compiles a guest basic block and then executes it
• At the end of each translated code section there is a jump back to HVX and the process repeats
Performance evaluation cont.
VT/SVM implementation in HVX

- HVX can expose VT enabled virtual hardware
- Can now run hypervisors like KVM, VMware ESXi, Hyper-V etc. in the cloud

Nested virtualization

Nested^2 virtualization
3. application framework

- Define an entire multi-VM/application end-to-end (RESTful API)

- Describe relationship between application components, external interfaces, configuration

- Every aspect of a Cloud Application can be coded

```python
def define_sugarcrm():
    # Define the SugarCRM Application, using constructs we supply
    # Define the VMs
    lb = VM(name = "LB", num_cpus = 1, memory = "8GB", image = "crm_lb_image",
            supplied_services = [Service(hostname="lb", port=80, global=True)],
            required_services = [Service(hostname="app1", port=80),
                                 Service(hostname="app2", port=80)])

    app1 = VM(name = "App1", num_cpus = 1, memory = "16GB", image = "crm_app_image",
              supplied_services = [Service(hostname="app1", port=80),
                                   Service(hostname="db")])

    app2 = VM(name = "App2", num_cpus = 1, memory = "16GB", image = "crm_app_image",
              supplied_services = [Service(hostname="app2", port=80),
                                   Service(hostname="db")])

    db = VM(name = "DB", num_cpus = 1, memory = "16GB", image = "crm_db_image",
            supplied_services = [Service(hostname="db")])

    # link the different services (can probably be done automatically)
    lb.required_services.app1 = app1.supplied_services.app1;
    lb.required_services.app2 = app2.supplied_services.app2;
    app1.required_services.db = db.supplied_services.db;
    app2.required_services.db = db.supplied_services.db;
    sugar = Application(name="SugarCRM", vms = [lb,app1,app2,db])

    # in the future we could also define runtime behavior here, SLAs, etc... for example:
    when(db.average_cpu(minutes=15) > 80).then(alert(email="git@ravello.com",
                                           subject='help', body='the db is over-loaded'))
    # in a similar way we can duplicate servers (for scale out), delete (for scale in), etc...
    return sugar;
```

Python bindings of Ravello’s RESTful API
putting it all together ...

infrastructure to encapsulate multi-VM applications **without any changes** and enable true hybrid deployments
Use cases
QUESTIONS?
thank you
Backup
Ravello capsule

- Ravello developed an application centric entity called **Capsule**.
- The Capsule comprise of everything that is required to run the app:
  - VM images
  - Overlay network
  - Overlay storage
  - Definitions of Capsule behavior under changing conditions
    - SLA, Constrains, Regulation
- Ravello have built an self-optimizing tool that maps the Capsule into any target cloud:
  - No changes are required for existing applications
  - Enables cost saving and performance optimizations
  - Assumptions: application is already virtualized
    - Any format is supported: ESX, KVM, HyperV, XEN
- Implementation is based on unique nested virtualization technology
  - Current “management tool” approach cannot really solve the problem.
- The VMs within the Capsule, connects to external services just like before.