Agenda

- PaaS today
- OSGi Cloud Ecosystems
- 'Demo'
PaaS offerings today (1)

• General deployment containers
  – Application Servers
    • i.e. deploy .WAR files in the cloud
  – Continuous Build Jenkins/Hudson

• Dynamic Language based
  – e.g. Python/Ruby/JavaScript
  – Often come with a Cloud-based IDE

• Support for specific technology
  – E.g. deploy Google App Engine on my own cloud
PaaS offerings today (2)

- Typically per cloud-VM instance
  - you need to do your own scaling
  - sometimes it can dynamically add replicas
    - of your entire deployment
  - instances don't know much about others
    - although they might connect to a cloud data store

- Today's PaaS
  - suitable if you can deploy multiple instances of the same app
  - harder to use if you have a composite application
    - with a variety of components
    - across different nodes
We need a better cloud!

- Where things are more dynamic
- Where a bunch of **different** nodes work together to form one logical application
- Where not every cloud VM image has just one purpose
- Where each cloud image could play different roles over time
- Where we can easily migrate from one cloud vendor to another
Example cloud-based Application (1)

- Has a web front-end
  - host needs to be accessible from outside
    - (well known hostname)
- Uses messaging
  - replicated message broker
- Has a data store
- And a back end
  - does the heavy lifting, computation
    - integrates with 3rd party providers
- Each entity on a different node
Example cloud-based Application (2)

Every entity runs on a different cloud node.

- **User interacts**
- **Submit request**
- **Message Queue**
  - Notifies
- **Web Front-end**
  - Store request data
- **Database**
  - Read and update request data
- **Back End**
Dynamic Scaling and correction

- I want all these elements to scale dynamically
  - possibly based on some domain-specific rules
- I want the system to be able to correct itself
  - e.g. when a Cloud VM dies/becomes inaccessible
- I don't want to worry about reconfiguring clients
  - when a service moves to another node
- I want the system to be portable
  - so I can move to another cloud
  - in case of entire cloud failure
How do I get all this?

I need an entity that orchestrates, a Provisioner

- Needs to be able to remotely deploy
  - to the nodes in the system
  - decide at runtime what the role of a certain VM is

- Needs to know the topology
  - as well as the application specifics

I need a flexible cloud platform

- repurpose a Cloud VM
- without sending the whole image to a machine again

Portability: I want to shield myself from cloud-vendor specific APIs

- as much as possible
- standards based
OSGi Cloud Ecosystems

• A number of OSGi frameworks
  - great because of their dynamic capabilities
• Each in a separate VM/cloud node
• Bound together to form a logical ecosystem
  - via ecosystem-wide discovery mechanism

• Environment
  - OSGi Frameworks can find out about other frameworks
  - Can easily communicate with other frameworks
  - and services running on these frameworks
A highly dynamic OSGi environment where Cloud nodes/VMs, bundles and services come and go.
How does this work?

- Start off with a number of 'empty' OSGi frameworks in the cloud
  - basic infrastructure provided
    - remote services capability
    - ecosystem support
  - no application logic
- A provisioner decides what goes where
  - knows about your application
  - and reacts to changes in topology / runtime system health
- Discovery component provides OSGi service visibility across frameworks
  - deals with service migrations
Ecosystem Support

- Each framework registers
  - `FrameworkStatus` service
  - Remote Deployment service
- These are visible across the Ecosystem
  - as OSGi services
  - (via OSGi Remote Services specs)
- But not outside
FrameworkStatus Service

• One per running Framework in the Ecosystem
  - Available via Remote Services
• Provides information about the framework

```java
public interface FrameworkStatus {
    String getFrameworkVariable(String name);
}
```

• Static metadata as Service Registration properties
• Dynamic metadata via getFrameworkVariable()

<table>
<thead>
<tr>
<th>static metadata</th>
<th>dynamic metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloud type/version</td>
<td>available memory</td>
</tr>
<tr>
<td>location (country / area)</td>
<td>available diskspace</td>
</tr>
<tr>
<td>OSGi framework version</td>
<td>machine load factor</td>
</tr>
<tr>
<td>processor architecture</td>
<td>network throughput</td>
</tr>
<tr>
<td>Java version</td>
<td>...</td>
</tr>
<tr>
<td>IP address</td>
<td></td>
</tr>
<tr>
<td>app-defined (custom)</td>
<td></td>
</tr>
</tbody>
</table>
Provisioning

- Need a Remote Deployment API
  - with Java client API
  - that spans the Ecosystem
  - works in the cloud
  - possibly an RFC 182 (REST)-based solution
A sample Provisioner (1/2)

// Listen for the FrameworkStatus service
ServiceTracker st = new ServiceTracker(context, FrameworkStatus.class.getName(), null) {
    public Object addingService(ServiceReference reference) {
        topologyChanged();
        return super.addingService(reference);
    }

    public void removedService(ServiceReference reference, Object service) {
        topologyChanged();
        super.removedService(reference, service);
    }
};

// React to changes in the topology
void topologyChanged() {
    // Deploy the web front-end
    if (getDeployments(WEB).size() == 0)
        addDeployment(WEB);

    // Deploy the back-end Service max twice.
    if (getDeployments(SERVICE).size() < 2)
        addDeployment(SERVICE);
}
void addDeployment(DeploymentType type) {
    ServiceReference[] possibleFrameworks = getFrameworkReferences();
    ServiceReference target = getMostSuitableFramework(type, possibleFrameworks);
    if (target == null) {
        // No suitable framework found
        return null;
    }

    String[] bundles = getDeploymentBundles(type, target);
    deployToFramework(target, bundles);
}

ServiceReference getMostSuitableFramework(DeploymentType type, ServiceReference... frameworks) {
    for (ServiceReference ref : frameworks) {
        if (type.equals(WEB)) {
            Object prop = ref.getProperty("...framework.ip");
            if (prop instanceof String)
                if (((String) prop).startsWith("web"))
                    return ref;
        } else if (type.equals(...)) {
            // ...
        }
    }
    return null;
}
How do my distributed business components interact?

- Develop them as OSGi Services
  - then use them via OSGi Remote Services...

- To publish a service in the ecosystem

  ```java
  MyService ms = new MyServiceImpl();
  Dictionary msProps = new Hashtable();
  msProps.put("service.exported.interfaces", ";*");
  msProps.put("service.exported.configs", "osgi.configtype.ecosystem");
  context.registerService(MyService.class.getName(), ms, msProps);
  ```

- Discovery component gives visibility across ecosystem
  - System will react automatically to services moving to other VMs/nodes
  - New instances appearing

- Client simply uses OSGi services as normal

  ```java
  MyService ms = ... // Remote Service from Service Registry
  ms.doit(); // Invoke Remote Service
  ```
Summary: OSGi Cloud Ecosystems

- A fluid system where deployments can come and go
- Provisioner decides what goes where
  - Cloud nodes get added, removed
    - Bundles might get deployed, moved
  - Ecosystem-wide services appear, disappear
    - 0, 1, many instances
  - Service consumers don't need to be reconfigured
    - react automatically to changes
    - handled by OSGi Remote Services + Discovery
- System can be repurposed
- Topic of OSGi RFC 183
  - early access draft available now
Cloud at the OSGi Alliance

- RFP 133 (Cloud Computing) accepted
  - http://www.osgi.org/bugzilla/show_bug.cgi?id=114

- RFC 182 REST API

- RFC 183 Cloud Ecosystems
  - Available as EA draft:
  - Being discussed at the EEG

- Starting: RFP 158 Distributed Event Admin
  - more RFPs/RFCs likely to come...
  - Interested? Join the discussion!
See it in action

- Pilot on Red Hat's OpenShift cloud
  - get your own free dev instances
- All source code available in github
- For more details see
  http://coderthoughts.blogspot.com
  (June/July/August 2012 posts)
Demo Setup

Provisioner rules:
• servlet needs to go on specific hostname
• exactly 1 testservice
  • not on servlet host

User interacts

- displays available frameworks
- invokes TestService
Demo Setup

Provisioner rules:
- servlet needs to go on specific hostname
- exactly 1 testservice
  - not on servlet host

User interacts

- displays available frameworks
- invokes TestService
Demo Setup

Web Front-end (Servlet)

- displays available frameworks
- invokes TestService

User interacts

Provisioner rules:
- servlet needs to go on specific hostname
- exactly 1 testservice
  - not on servlet host

Back End
TestService

Back End
TestService
Demo Setup – The Servlet

**FrameworkStatus and TestService**

from Service Registry

- lists available frameworks

```java
void printFrameworks(PrintWriter out) {
    // frameworkRefs list all FrameworkStatus services from the OSGi Service Registry
    for (ServiceReference ref : frameworkRefs) {
        for (String key : ref.getPropertyKeys()) {
            out.println(key + " - " + ref.getProperty(key));
        }
    }
}
```

- invokes a 'computation' service

```java
void printTestService(PrintWriter out) {
    out.println("<H2>TestService invocation</H2>"ecaft);
    TestService svc = ... // from the OSGi Service Registry
    if (svc != null)
        out.println("TestService Result: " + svc.doit());
    }
```
Demo Setup – Services

FrameworkStatus is provided by the Ecosystem

TestService is a simple OSGi Service

```java
public interface TestService {
    String doit();
}
```

Registered with Remote Service props

```java
public class Activator implements BundleActivator {
    public void start(BundleContext context) throws Exception {
        TestService ts = new TestServiceImpl();
        Dictionary tsProps = new Hashtable();
        tsProps.put("service.exported.interfaces", "*");
        tsProps.put("service.exported.configs", "osgi.configtype.ecosystem");
        context.registerService(TestService.class.getName(), ts, tsProps);
    }
}
```
Demo

• Real demo available online:
  http://www.youtube.com/watch?v=8-9deWm67w0

• Following is a set of screenshots that capture the essence of the demo...
Dynamic Provisioning OSGi Ecosystem

In my recent post I looked at the OSGi as a plugin ecosystem. In this post I'm starting to look at an alternative, dynamic provisioning, where the OSGi framework is used to dynamically provision and re-provision components.

In my previous post I was using a static framework to start out with a number of identified components. I wrote a very simple demo that takes a list of components and provisioned them. When the framework goes down, it re-reprovisions services to an alternative.

You’ll see that, as in the previous post, the components are located. They are all part of the compliant Discovery mechanism. I wrote a very simple demo that connects to a list of components and provisioned them. When the framework goes down, it re-reprovisions services to an alternative.
remote: INFO: creating server for interface org.coderthoughts.cloud.provisioning.api.RemoteDeployer
remote: INFO: found handler for org.apache.cxf.dosgi.dw.service.RemoteServiceAdminCoreService
remote: INFO: Successfully registered CXF DOSGi servlet at /s2
remote: INFO: Could not find intent map file /OSGI-INF/intents/intent-map.xml
remote: INFO: Injected intents: {}
remote: INFO: Injected intents: [addressing-org.apache.cxf.ws.policy.WSPolicyFeature@724f83b4, logging-org.apache.cxf.feature.LoggingFeature@8424ef9d6, soap-org.apache.cxf.binding.soap.SoapBindingConfiguration@833967e, SOAP.1_1-org.apache.cxf.f.com.SoapBindingConfiguration@833967e, SOAP.1_2-org.apache.cxf.binding.soap.SoapBindingConfiguration@4def3666, HTTPPPROVIDED]
remote: INFO: Setting the server's publish address to be /remote: INFO: created server for interface org.coderthoughts.cloud.provisioning.api.RemoteDeployer
remote: INFO: Unhandled event type received: 2
remote: INFO: Zookeeper callback on node: /osgi/service_regiots.org.coderthoughts.cloud/provisioning/api/RemoteDeployer
Frameworks in the Cloud Ecosystem

OSGi Framework (local) - Free Memory: 163587 (kilo bytes) OSGi Framework (remote) - Free Memory: 149569 (kilo bytes)

TestService invocation
web
Framework: diy-0.1
Creation: 2012-09-09T14:00:11-04:00
UUID: f171a1cefd84657b4653d3683c96261
Git URL: ssh://f171a1cefd84657b4653d3683c96261@web-davidosgi.rhcloud.com/
~/.git/web.git/
Public URL: http://web-davidosgi.rhcloud.com/
Embedded: None

osg1
Framework: diy-0.1
Creation: 2012-09-09T14:03:06-04:00
UUID: 5c0c142f3b84dcc840f1034c2ffbf
Git URL: ssh://5c0c142f3b84dcc840f1034c2ffbf@osg1-davidosgi.rhcloud.com/
~/.git/osg1.git/
Public URL: http://osg1-davidosgi.rhcloud.com/
Embedded: None

osg12
Framework: diy-0.1
Creation: 2012-09-09T14:06:18-04:00
UUID: 1377f67d578bb6baa7b7187213e0a
Git URL: ssh://1377f67d578bb6baa7b7187213e0a@osg12-davidosgi.rhcloud.com/
~/.git/osg12.git/
Public URL: http://osg12-davidosgi.rhcloud.com/
Embedded: None

davidpop ~/clones $ rhc app start -a web -l davidredhat.com
Password:
RESULT: Success

davidpop ~/clones $
davidpop ~/clones $ rhc app start -a osg1 -l davidredhat.com
Password:
RESULT: Success

davidpop ~/clones $
davidpop ~/clones $ rhc app start -a osg12 -l davidredhat.com
Password:
RESULT: Success

davidpop ~/clones $
Frameworks in the Cloud Ecosystem

OSGi Framework (local) - Free Memory: 136123 (kilo bytes)
org.coderthoughts.framework.ip
org.osgi.framework.uid
googlesamples

OSGi Framework (remote) - Free Memory: 117422 (kilo bytes)
web-davidosgi.rhcloud.com
org.coderthoughts.framework.ip
org.osgi.framework.uid

OSGi Framework (remote) - Free Memory: 105060 (kilo bytes)
provisioner-davidosgi.rhcloud.com
org.coderthoughts.framework.ip
org.osgi.framework.uid

OSGi Framework (remote) - Free Memory: 77337 (kilo bytes)
org.coderthoughts.framework.ip
org.osgi.framework.uid

TestService invocation

- TestService (remote) status: OK Result: My Framework UUID is: 10829f30-b5fa-0011-1a38-9c32da7f1f46
remote: INFO: Unhandled event type received: 1
Frameworks in the Cloud Ecosystem

<table>
<thead>
<tr>
<th>OSGi Framework (local) - Free Memory: 122147 (kilo bytes)</th>
<th>OSGi Framework (remote) - Free Memory: 110472 (kilo bytes)</th>
<th>OSGi Framework (remote) - Free Memory: 164334 (kilo bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.coderthoughts.framework.ip web-davidosgi.rhcloud.com</td>
<td>org.coderthoughts.framework.ip provisioner-davidosgi.rhcloud.com</td>
<td>org.coderthoughts.framework.ip osgi2-davidosgi.rhcloud.com</td>
</tr>
<tr>
<td>org.osgi.framework.uuid a0a411d0-b4fa-0011-128d-fc85f00a6f6a</td>
<td>org.osgi.framework.uuid 40bc8d6a-affa-0011-18ad-8939fb7ca472</td>
<td>org.osgi.framework.uuid a0c3a2783-b5fa-0011-18ad-b4c460a544f2</td>
</tr>
</tbody>
</table>

TestService invocation

- TestService (remote) status: OK Result: My Framework UUID is: a0c3a2783-b5fa-0011-18ad-b4c460a544f2
Questions?
Acknowledgements

- Cloud image Wikimedia Commons
  - Stratocumuli by de:Benutzer:LivingShadow