Creating a Java™ Platform, Enterprise Edition (Java EE Platform) Appliance Using GlassFish™ and OpenSolaris™

Peter Karlsson, Technology Evangelist

Lee Chuk Munn, Staff Engineer

Java

Sun microsystems

Learn how we build a JavaEE application appliance

- Under funded
- Under pressure

for One Appliance per Geek™ (OAPG) cause!

GOAL

Agenda

- OpenSolaris™ Project / GlassFish Appliance
- OpenSolaris Project
  - Short Introduction to Solaris ZFS
  - (Another) Short Introduction to SMF
- GlassFish Application Server
- Appliance Features Overview
  - Installing Applications
  - Integration with SMF
- Summary

OpenSolaris/GlassFish Appliance

What is an Appliance?

- An instrument or device designed for a particular use or function
  - A solution to solving problem by enabling the proper function
- Typically hardware and software bundled and tightly integrated
- Example of appliance
  - Toaster
  - Wireless router

Criteria for an Appliance*

- Purpose
  - Scope – scope of operation
  - Design – native or packaged
  - Ends not means – what it achieves rather than how it is done
- Convenience
  - Ease of use – aim for plug-and-play
  - Simplicity – if you need to open it up, then it is not simple
  - Compatibility – harmony with the environment
- Value
  - Alignment – align to the business requirements
  - Cost – lower TCO and should cost less than sum of parts
  - Performance – does not only refer to speed but for achieving its purpose

* From http://www.b-eye-network.com/view/5790

Why a JavaEE Appliance?

- SMEs do not have the expertise to install and manage an appserver
  - Do not really care about the system, just what it can do for them
- Small to medium JavaEE application tend to have similar hardware and software requirements
  - Cater to these needs
- Telcos, hosting company deploy large number of these
  - Fairly generic
  - Should be manageable as a unit, not as independent OS and application server
- Application specific JavaEE appliance
  - Software vendor can leverage a generic appliance for OEMs
  - Think Karaoke appliance (maybe not)

JavaEE application as an Operating Environment

- Grown into a full fledged environment
  - Can write application without know anything about the underlying OS
  - Handles its own resources, security, connections, cache/pools, etc
  - Meta environment for managing the application server and Java Virtual Machine (JVM™)
  - Clustrable, *able words
- Grown in complexity and sophistication
  - Micro managing application servers when the environment only run a JavaEE application server
    - A bit like an extremely high end Hi-Fi kit
  - Should have a tight binding with the operating system
- Abstraction is higher, at component level
  - Solaris system/Linux/Unix file based abstraction

What Do We Want from the JavaEE Appliance?

- Easy setup
  - Close to zero touch
  - LCD for system display and configuration
  - Flashing LEDs for status
  - Easy deployment
- Low cost
  - Off the shelf parts
  - Free and open source software
- 'Acceptable' performance
- Operable in room temperature
  - 'Normal' power supply – 110V or 230V
- Auto update, if permission is given
- Robust – install and forget should be the motto

What are the Appliance Non-Goals

- Target for deployment not for development
  - Can certainly use it for development
- Not intended to run resources
  - Database
  - Mail services
  - Message queues
- Set SPECjbb benchmarks
- Running any other applications besides what comes preloaded

Challenges for a JavaEE Appliance

- Application Server is a complex beast
  - Challenge is how to make it usable as an appliance?
- Learn from Grails and Rails
  - Provide excellent support for the most common use cases, workflow
  - Conventions over configuration
- View application as a unit rather than sum of parts
  - Eg. web application, EJB™ architectures, database
  - Currently difficult without proprietary extensions in deployment descriptor
- Aim at users who know a *little* about web application
- Reset to “factory installed” if all else fails

OpenSolaris Project as the Base

- Lots of useful features to support 'appliancing' GlassFish application server
  - Solaris ZFS for preserving state, versioning, backups
  - SMF (Service Management Framework) for service management
  - Containers for security
  - Mature and robust
- Can easily be customized
  - Hardening
  - Small footprint
- Robust and tested
- Familiarity

Major Software Components

- OpenSolaris source code 2008.05
  - Leverage the OS facility to simplify deployment and management of JavaEE applications
  - Solaris ZFS, Containers, SMF
- JavaSE platform 6
  - Latest and greatest
- GlassFish application server
  - Open source
  - Fairly popular and easy to setup
  - Standards compliant
- Groovy and Grails
  - Tight integration with the JVM software
  - Use as a glue for tying the system
  - Grails is used to develop the appliance's management interface

Major Hardware Component Prototype

- Solaris platform friendly motherboard
  - Intel D201GLY2 Mini-ITX
    - Soldered down Intel® Celeron 220 with a 533 MHz system bus
    - 1GB RAM
    - 8GB CF as system disk
- picoLCD 20x2 LCD display
- Keypad
- 12V external power supply

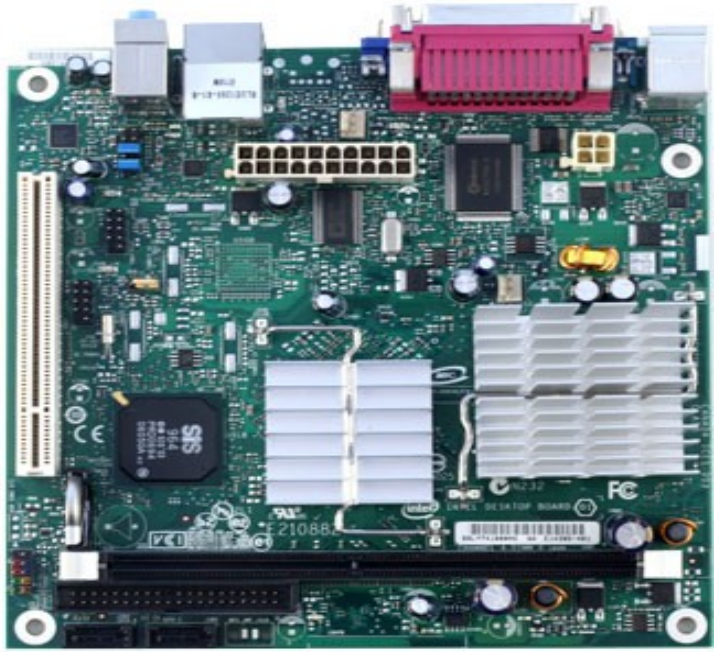


Image from http://www.mini-box.com/M200-LCD-Enclosure?sc=8&category=87

Image from http://www.mini-box.com/picoLCD-20x2-OEM

OpenSolaris Project

Miniaturizing OpenSolaris Project

- Only install the bare minimum of what we will need
  - Anything else can be downloaded later from the OpenSolaris project network repository automatically as we need it
- Only services that we really need will be enabled

Open Solaris project configuration

- Use LCD and keypad to do basic configuration
  - Set up ip-address, static or DHCP
  - Hostname
- Auto detect JavaEE applications on USB drive
  - Detect and deploy applications from USB drive to local media
- Use SMF to manage JavaEE applications
  - Auto start, define dependency on resources, etc
- What I did
  - Wrote a driver to interface with the picoLCD and using the keypad as input
  - Used standard OpenSolaris source code configuration files and APIs
  - Integration with the OpenSolaris source code removable media manager “tarmac” to detect USB events
  - Integrated GlassFish application server with SMF

Short Introduction to Solaris ZFS

Revolutionary Solaris ZFS File System

Best File System

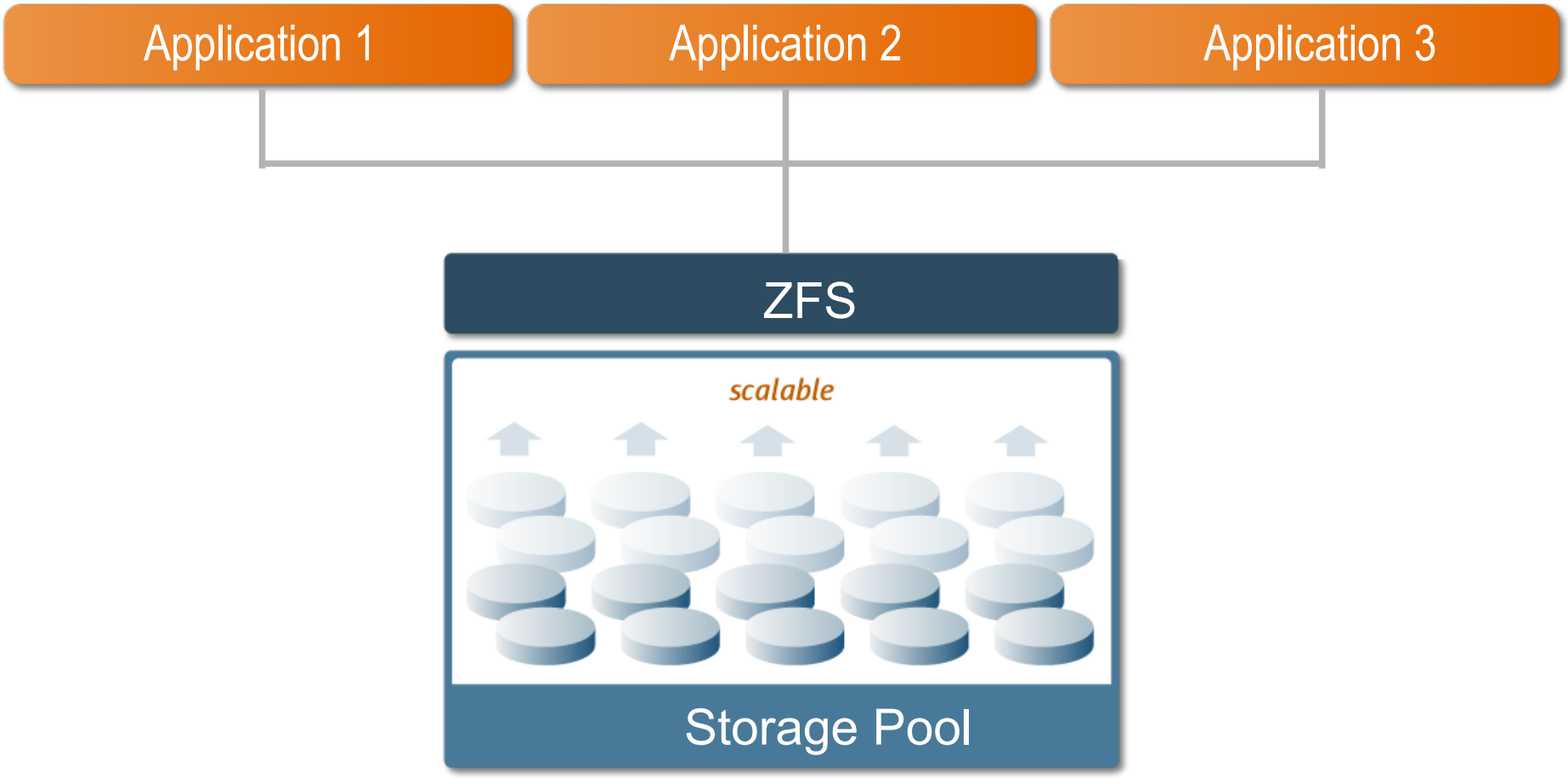
- End-to-end data integrity
  - Copy-on-write transactions
  - 64-bit checksums
- Simplified administration
  - Storage pools
  - No slices, volumes, partitions
- Infinitely scalable
- Huge performance gains

Simple, Reliable, Scalable

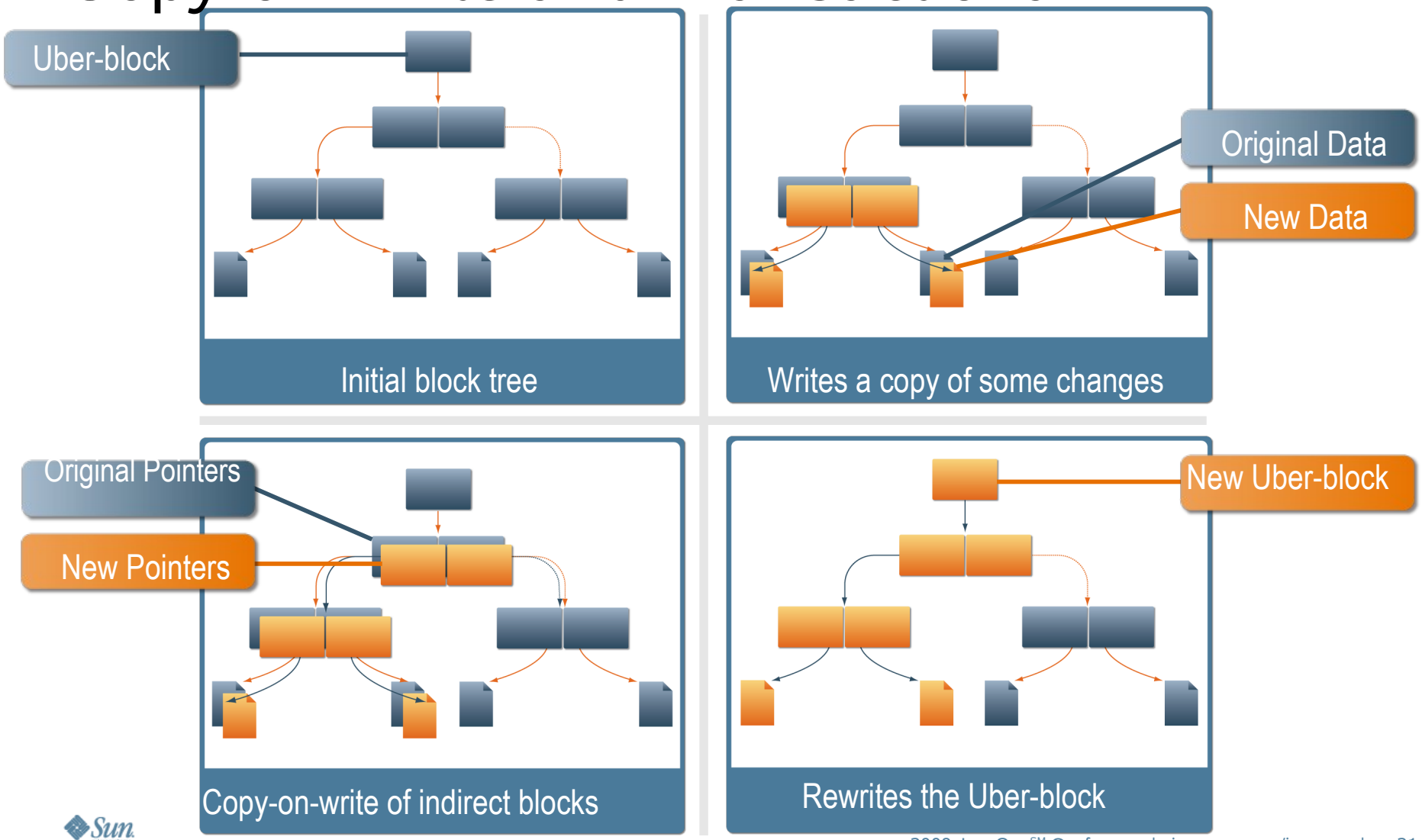
ZFS is a 128-bit file system, so it can store 18 billion billion times more data than current 64-bit systems.

No More Volume Manager!

Automatically add capacity to shared storage pool

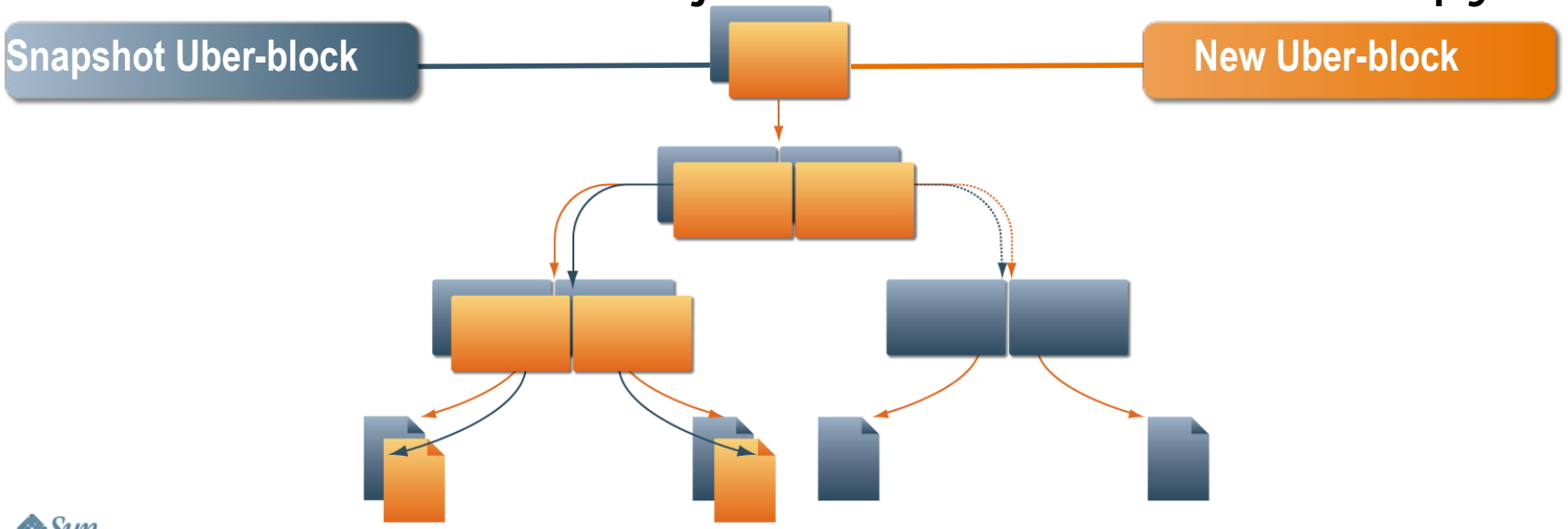


Copy-on-Write and Transactional



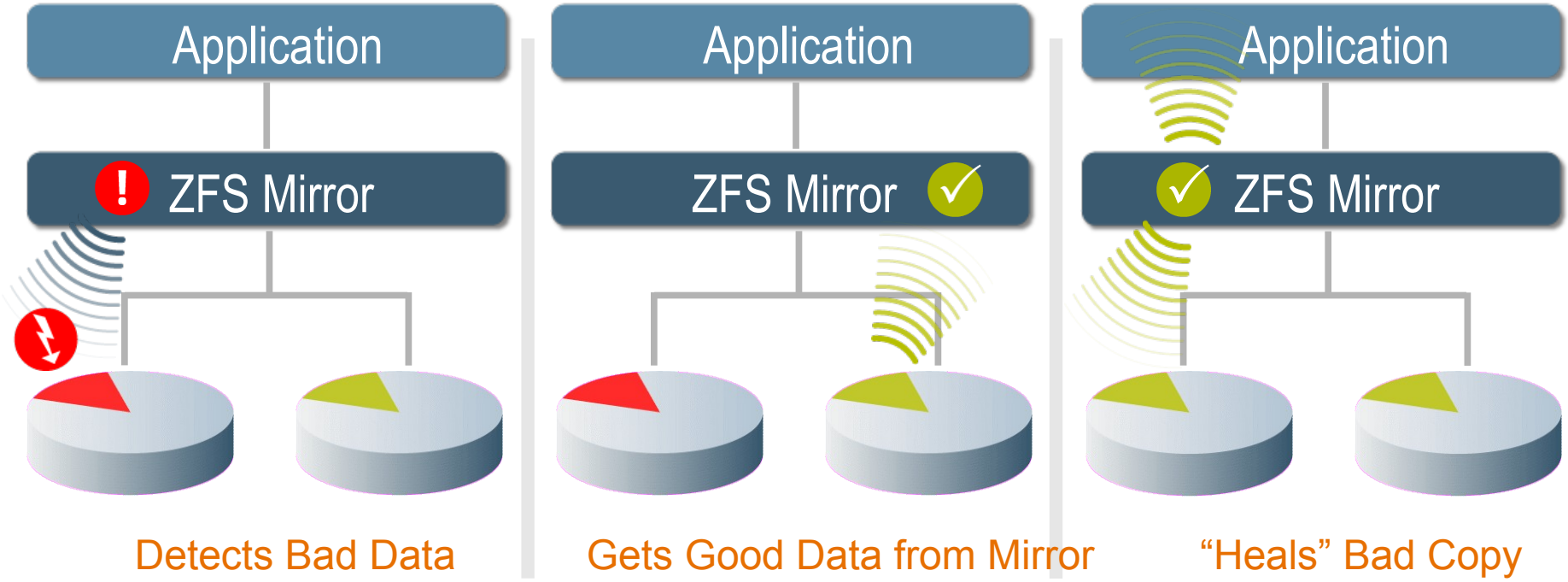
Solaris ZFS Snapshots

- Provide a read-only point-in-time copy of file system
- Copy-on-write makes them essentially “free”
- Very space efficient – only changes are tracked
- And instantaneous – just doesn't delete the copy



Self-Healing Data

Solaris ZFS can detect bad data using checksums and "heal" the data using its mirrored copy.



How we use Solaris ZFS in our project

- Keep a “factory” configuration snapshot for easy reset
  # zfs rollback -r root_pool@factory
  - Resets configuration to “factory default”
- Each application resides on its own Solaris ZFS file system
  - Allows for quick backup and recovery using snapshots
  - By using Solaris ZFS send/receive we could implement remote storage of backups
- We can enable use of “ditTo blocks” for important data
  - Allow for data redundancy even though we only use 1 drive
  - Protect against partial failure of storage media

Short Introduction to SMF - Service Management Framework

SMF – Short Introduction

- Basic idea, manage services not processes
  - Users care about the service
- Describe your service in an XML manifest
  - What's the name of your service
  - What services do you depend on
  - How do you start, stop and restart the service
  - Actions to be taken in case of various failure scenarios, full, partial or failure of other service that I depend on
  - Where to find more information about my service, log files etc.

GlassFish Application Server

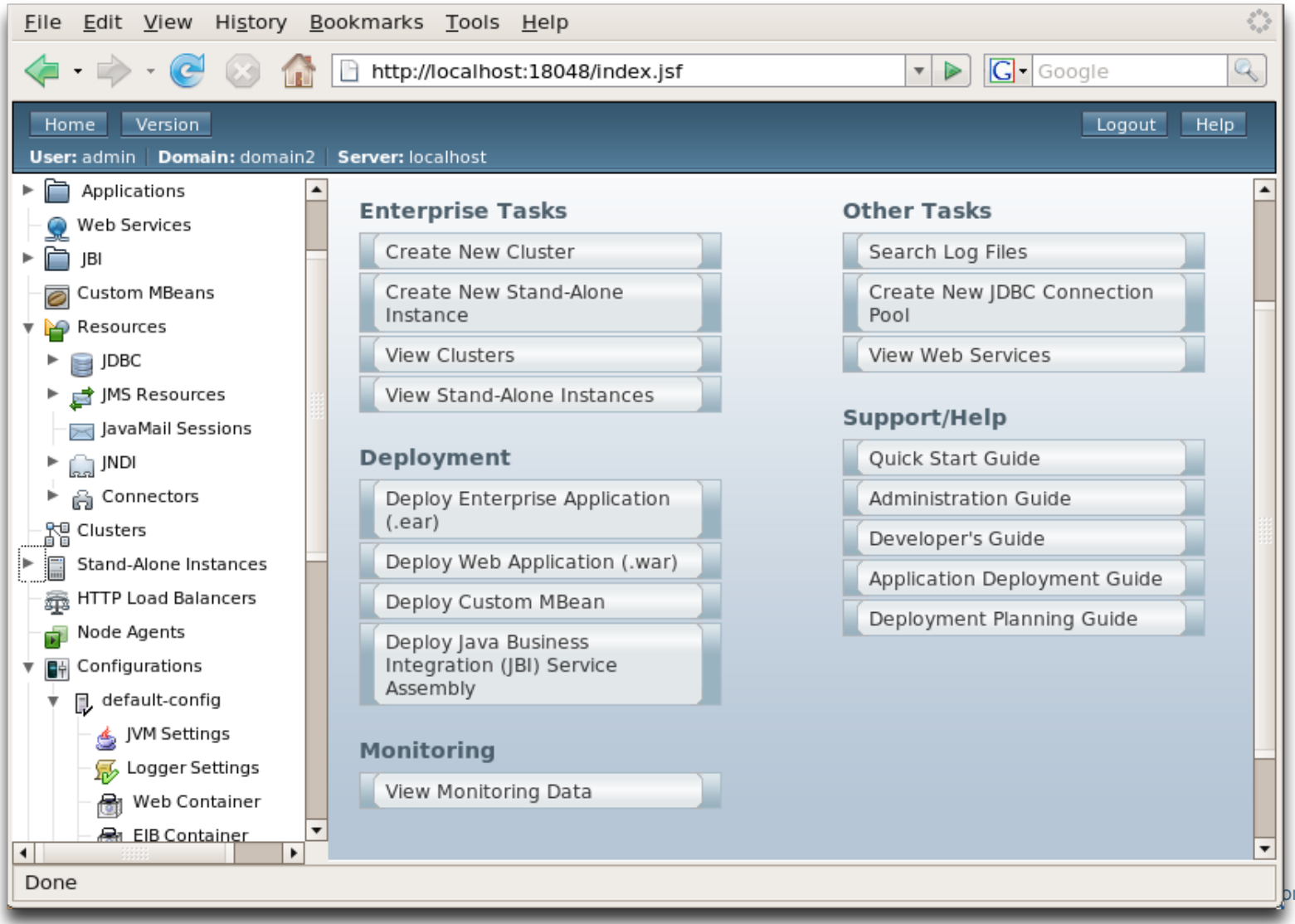
GlassFish Application Server V2 Features – 1

- JavaEE platform 5 compliant
- Metro Web Service Stack
  - Performance, advance WS features and Microsoft interoperability
- Clustering, load-balancing and HA
  - Unified management
- Web tier
  - Grizzly, dynamic web container, fast JSP™ framework compilation
- Java Business Integration (JBI) support
- Observability
  - Graphical, command line tools
  - JMX software based
  - Call flow, self-management
  - Multi tier provisioning with N1 SPS

GlassFish Application Server V2 Features – 2

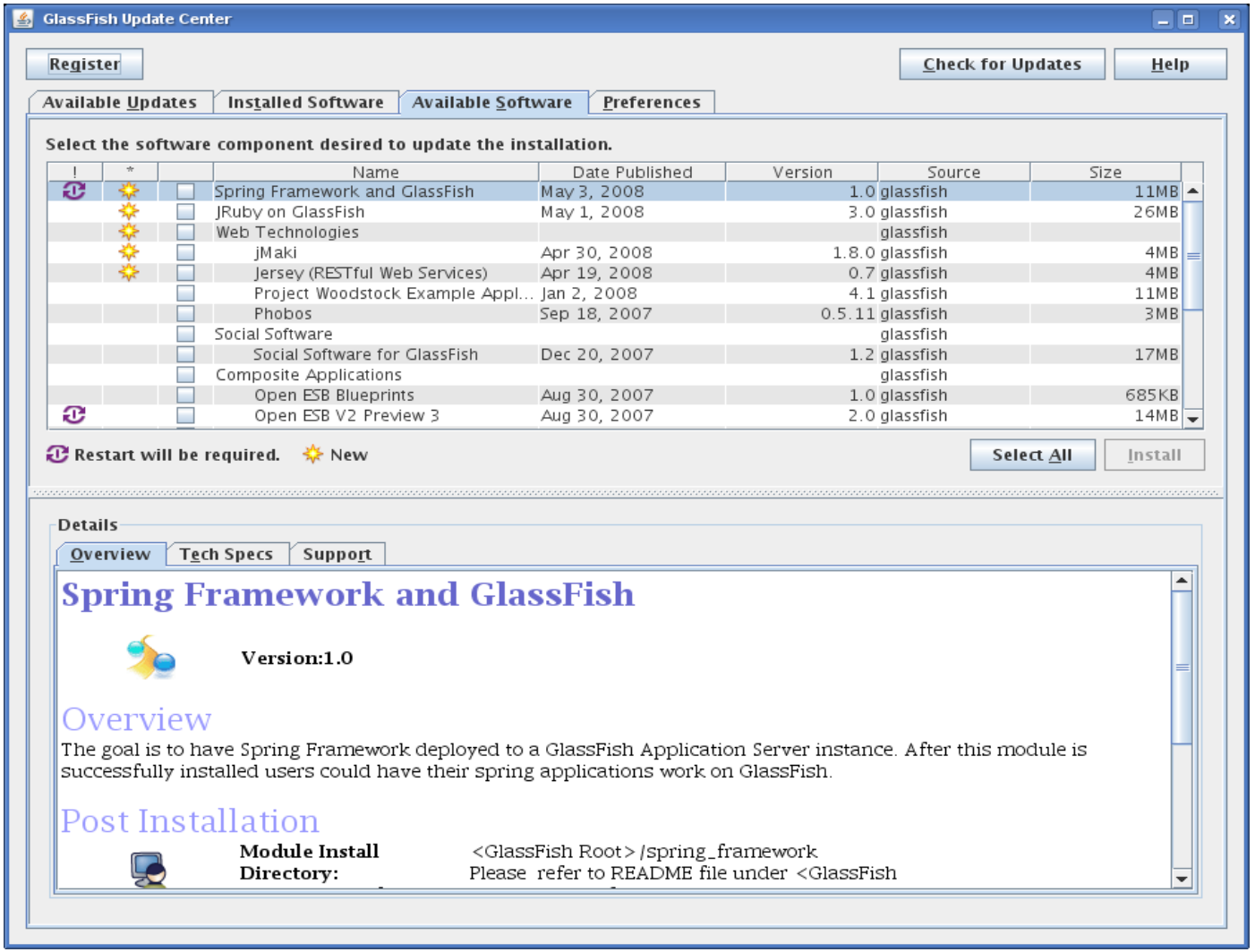
- Multiple user profiles
  - Developer, cluster, enterprise
  - Upgrade from one to the other
- Better startup time
  - Comparable to Tomcat
- Latest Web 2.0 and cool technologies
  - Comet, Ruby on Rails, jMaki, SIP
- Update center
  - Provision and install new features, frameworks, etc
- Tools support
  - NetBeans™ software, (My)Eclipse, IntellJ

Screenshot of Web Admin

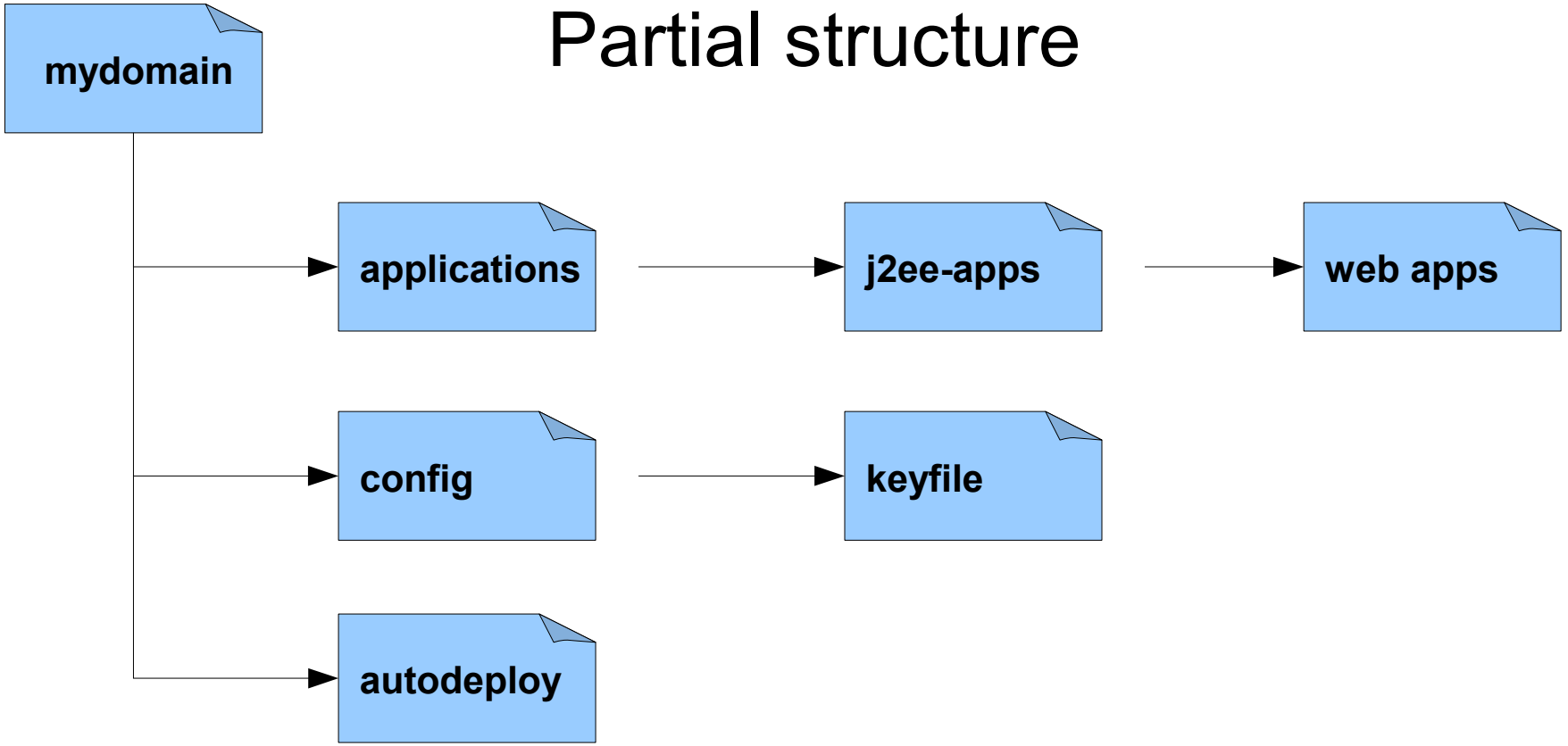


Screenshot of VisualVM with GlassFish Application Server

Screenshot of Undate Center



GlassFish Application Server Domain Directory Structure




```
asadmin start-domain –domaindir /path/to/domain mydomain
```

Appliance Features Overview

Appliance Startup

- Connect LAN cable to hub
- Power on
- Set host name
- Set static IP address or use DHCP
- Point your browser to http://server.com:8080
- This is the manual

Web Applications As First Class Citizens

- Current version of appliance makes it extremely easy to install and manage web applications
- Web applications includes the following
  - Traditional web applications JSP framework/ Servlet
  - JSF applications
  - AJAX based application
  - Dynamic languages based web applications
    - As long as it can run on Servlet/JVM software combo
- If you can create a WAR file, should be able to deploy
- Other application types are not supported through the simplified interface
  - Go to GlassFish application server admin console
  - http://localhost:4848

Administration Features

- Administration of appliance is done via Grails application
  - Meant for simple, common use cases
  - For custom configurations, still need to go back to admin console
- The following can be configured
  - Application
  - User
  - Network
  - Resources
- All these can be perform via the LCD panel
- Does not include cluster
  - Telnet in to appliance if you really want to do this
- Features not advertised is accessible if you know how
  - Will eventually provide a way to shut these off
  - Harden the appliance – Solaris application and GlassFish application server

Resources

- Uses existing asadmin command to do this
  - create-jdbc-connection-pool
  - create-jdbc-resource
  - javamail-resource
  - create-persistence-resource
- Prepackage common resources to reduce settings
  - Make it really easy to define data sources – lessons from Grails/Rails
  - Connection pool and JDBC™ API resource – Oracle, Derby/JavaDB, MySQL, Postgres
  - Persistence resource

Users

- Appliance defaults to 'File Realm' security
  - GlassFish application server supports File, Certificate, JDBC API and Custom
  - Simplest and easiest to understand: username / password
- Simplified interface to create users
- Do not envisage target users to tie into LDAP, Custom, etc.
- Users here are GlassFish application server users not OpenSolaris source code
  - Do not allow adding users to OpenSolaris source code
  - Users are not encourage to run any other application on appliance

Installing Application

Scenarios

- Deploy JavaEE application: USB Drive
- Deploy JavaEE application: URL
- Upgrade JavaEE application: USB/URL

- Backup
  - JavaEE application
  - System backup

- Platform upgrade

- Reset to “factory” configuration

Add a JavaEE application: USB drive

- Vold detects USB drive inserted in USB slot
- Scan USB drive for war files
- Copy war file to system
- Create Solaris ZFS for application
- Create SMF manifest for application, import to SMF
- Tell GlassFish application server to deploy application on the created Solaris ZFS
- Take “deploy” snapshot of Solaris ZFS filesystem
  - **zfs snapshot app/”appname”@deploy**
- Tell SMF to “enable” application
- And we're up and running

Add a JavaEE application: URL location

- Select “Add Application” from Menu on LCD
- Enter URL using the keypad
- Use wget to download the war file to local system
- Create Solaris ZFS for application
- Create SMF manifest for application, import to SMF
- Tell GlassFish application server to deploy application on the created Solaris ZFS
- Take “deploy” snapshot of Solaris ZFS filesystem
- Tell SMF to “enable” application
  - SMF tells GlassFish application server to start application

Upgrade application : USB/URL location

- Select upgrade application on LCD menu
- Select application to upgrade
- Select USB or URL deployment
- The appliance SW does the following steps
  - Copy new war file to system
  - Tell SMF to temporarily “disable” application
  - Take “pre upgrade” Solaris ZFS snapshot
  - Install web application
    - Application will be reinstalled if exist
  - Take “post upgrade” Solaris ZFS snapshot
    - Rename “pre upgrade” Solaris ZFS snapshot” to “appname@current-1”
    - Rename “post upgarde” Solaris ZFS snapshot to “appname@current”
  - Tell SMF to “enable” the application

Upgrade: Rollback to previous version

- Select “rollback application” on LCD menu or web interface
- Select application to “rollback”
- Select version to “rollback” to
  - We keep a history of 3 versions
- The appliance SW does the following steps
  - Tell SMF to temporarily “disable” the application
  - Do a Solaris ZFS rollback to selected version
    - Solaris ZFS rollback app/appname@current-1
  - Rename Solaris ZFS snapshot to reflect current setup
    - @current -> @current+1
    - @current-1 -> @current
  - Tell SMF to re-enable the application
- If OK select commit on menu
  - All Solaris ZFS snapshots later than @current gets destroyed

Backup JavaEE application

- Select Backup application on LCD
- If you have more than one application deployed, select application to backup or ALL

- For each application:
  - SMF temporarily “disables” the application
  - Take Solaris ZFS snapshot of application
  - SMF “enables” application
- Option to save backup on remote media
  - USB
  - URI (nfs/ftp)

Backup System

- Select “Backup” -> “System” on LCD menu
- SMF temporarily “disables” GlassFish application server
  - As all applications “depend” on GlassFish application server they will “disabled” as well
- Take Solaris ZFS snapshots
  - All applications
  - GlassFish application server
  - OpenSolaris source code
- SMF “enables” GlassFish application server and applications
- Option to save backup on remote media
  - USB drive
  - URI (nfs/ftp)

“Factory reset”

- Select “System” -> “Reset to default configuration” on LCD
  OR
- Push “RESET” button

- SMF “disables” applications and GlassFish application server
- Destroy Solaris ZFS filesystems for installed applications
- Rollback Solaris ZFS for GlassFish application server and OS to @install snapshot
- Reboot

Integration with SMF

Using SMF

- SMF manifest will be automatically generated whenever an applications or resources are installed
  - Generated with Groovy Builders
- Use SMF to create dependencies between application and resources
  - Define a SMF when ever a resource is created – eg. connection pool
  - Define a SMF dependency from application to resource whenever an application that uses that resource is installed
- Check all dependencies when we start an application
  - Done by OpenSolaris project through the appliance user interface
- Other things we can do
  - Automatically restart an application or resource when either one dies
  - Automatically start/shutdown application and resources whenever appliance startup/shutdown

```
<?xml version='1.0'?>
<!DOCTYPE service_bundle SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<service_bundle type='manifest' name='export'>
<service name='application/database/mysql' type='service' version='0'>
   <dependency name='network' grouping='require_all' restart_on='none' type='service'>
      <service_fmri value='svc:/milestone/network:default'/>
   </dependency>
   <dependency name='filesystem-local' grouping='require_all' restart_on='none' type='service'>
      <service_fmri value='svc:/system/filesystem/local:default'/>
   </dependency>
   <exec_method name='start' type='method' exec='/lib/svc/method/mysql start' timeout_seconds='60'>
      <method_context/>
   </exec_method>
   <exec_method name='stop' type='method' exec='/lib/svc/method/mysql stop' timeout_seconds='60'>
      <method_context/>
   </exec_method>
   <instance name='dev_db' enabled='false'>
      <method_context project=':default' resource_pool=':default' working_directory=':default'>
         <method_credential group='mysql' limit_privileges=':default' privileges=':default' supp_groups=':default' user='mysql'/
      </method_context>
      <property_group name='mysql' type='application'>
         <propval name='bin' type='astring' value='/usr/mysql/5.0/bin'/>
         <propval name='data' type='astring' value='/export/db_data/dev_db'/>
         <propval name='port' type='astring' value='3308'/>
      </property_group>
      <property_group name='general' type='framework'>
         <propval name='action_authorization' type='astring' value='solaris.smf.manage.mysql/dev_db'/>
         <propval name='value_authorization' type='astring' value='solaris.smf.manage.mysql/dev_db'/>
      </property_group>
   </instance>
```

```
    <instance name='prod_db' enabled='false'>
      <method_context project=':default' resource_pool=':default' working_directory=':default'>
        <method_credential group='mysql' limit_privileges=':default' privileges=':default'
                                                    supp_groups=':default' user='mysql'/>
      </method_context>
      <property_group name='mysql' type='application'>
        <propval name='bin' type='astring' value='/usr/mysql/5.0/bin'/>
        <propval name='data' type='astring' value='/export/db_data/prod_db'/>
        <propval name='port' type='astring' value='3306'/>
      </property_group>
      <property_group name='general' type='framework'>
        <propval name='action_authorization' type='astring'
                          value='solaris.smf.manage.mysql/prod_db'/>
        <propval name='value_authorization' type='astring'
                           value='solaris.smf.manage.mysql/prod_db'/>
      </property_group>
    </instance>
    <stability value='Evolving'/>
    <template>
      <common_name>
        <loctext xml:lang='C'>MySQL RDBMS</loctext>
      </common_name>
      <documentation>
        <manpage title='MySQL 5.0.45' section='1'/>
        <doc_link name='mysql.com' uri='http://dev.mysql.com/docs'/>
      </documentation>
    </template>
  </service>
  </service_bundle>
```

Summary

Lessons Learnt

- Never attempt to build an appliance 4 weeks before delivery!
- Deciding what type of application to support
  - Web application is a really simple decision
  - Make it real easy to deploy Grails and Rails application
    - Rails config/database.yml
    - Grails conf/DataSource.groovy
- Grails is excellent if we were working from a database
  - Cannot leverage Grails fully
  - Still quite productive
  - Copy GSP from existing Grails app
  - Think like a script dude instead of a Java guy – short cuts are okay

Next Steps

- Self configuration JavaEE application package
  - Resources are defined in application package – eg. sun-web.xml
  - Install resource along with application
  - Excellent if used with 'cloud' computing
    - Eg. Amazon S3 or SimpleDB
- Tools support
  - Use IDE to configure deployment descriptor for the appliance
  - Currently we are not reading sun-web.xml file
  - Can get better understanding of application
- Cluster/HA support
  - GlassFish application server has near zero touch for 'memory' cluster configuration
  - Appliance should leverage that

Acknowledgment

- Jason Huang – helped with the code for managing GlassFish application server

THANK YOU

Peter Karlsson, Technology Evangelist
Lee Chuk Munn, Staff Engineer

Speaker's logo here (optional)