MODIFYING TRADITIONAL APPLICATIONS FOR MORE COST EFFECTIVE DEPLOYMENT

Eric H. Nielsen Ph.D.
VP SaaS Platform Architecture
CA Technologies
e.h.nielsen @ ieee.org

IEEE Software Technology Conference
STC 2014, Long Beach, CA USA April 2, 2014

abstract

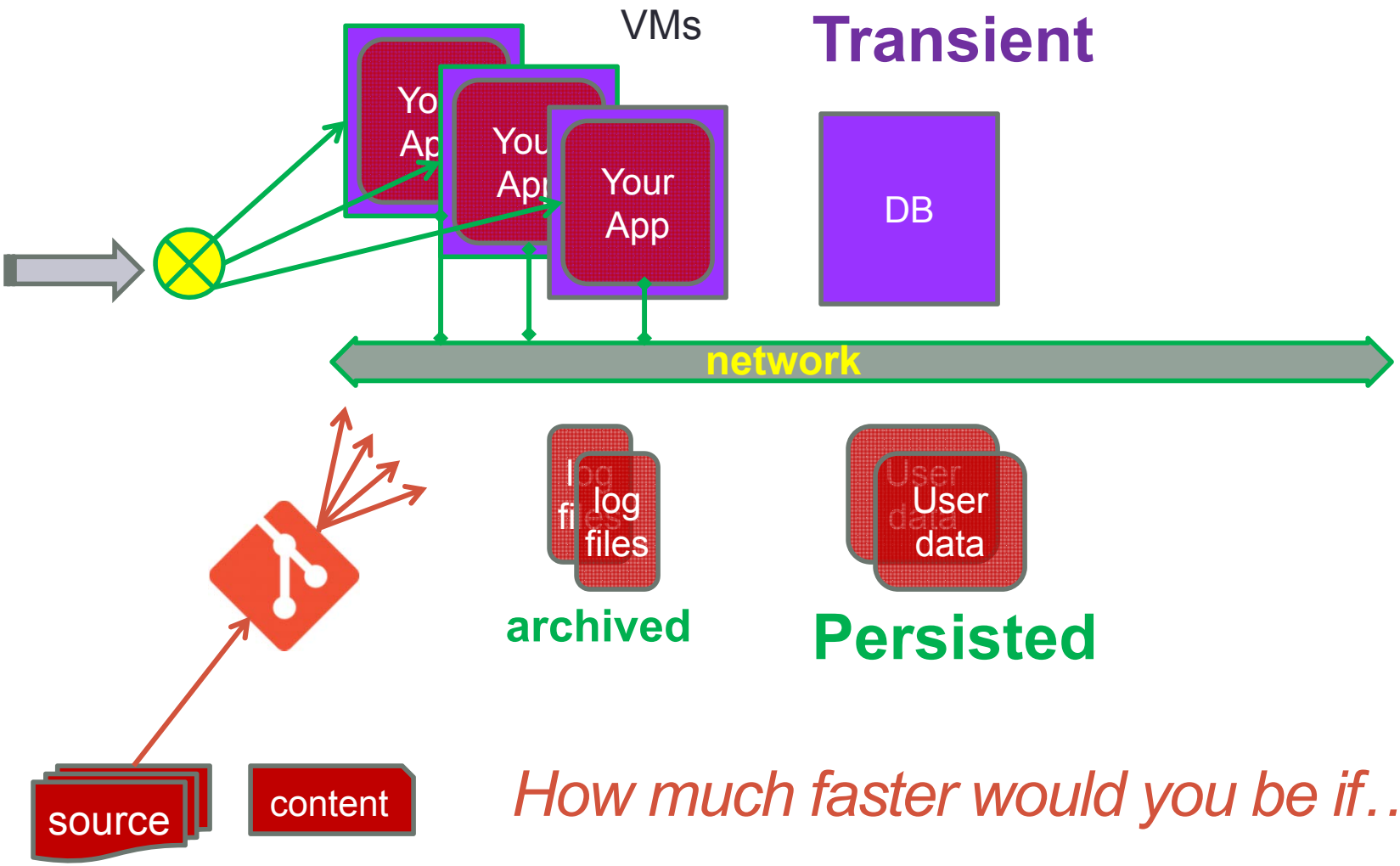
Traditional business applications will not compete with green field multi-tenant SaaS applications in terms of cost or usability. Still, service providers can provide value in deploying these traditional business applications in the cloud if the application is modified to be more cloud-friendly. In order to make a significant impact on the cost of operation, the application must be tuned to allow use of simpler scalable SaaS-oriented automation for management and operations.

The most significant operational improvement for a traditional application is gained through the physical separation of its configuration and persistent user data from its execution resources. Traditionally applications have assumed installation in a single location for their lifetime. This allows persistent data and configuration to be intertwined in files and databases, and leads to idiosyncratic maintenance.

Decoupling application code, application configuration, and user data typically requires resolving a laundry list of details. Once complete, the application configuration is recreated fresh on every startup, and then remarried to its persistent data. This level of statelessness enables operational maintenance procedures to become simpler and more uniform, and occur with less downtime. This results in reduced cost of ownership, as well as enhanced security.

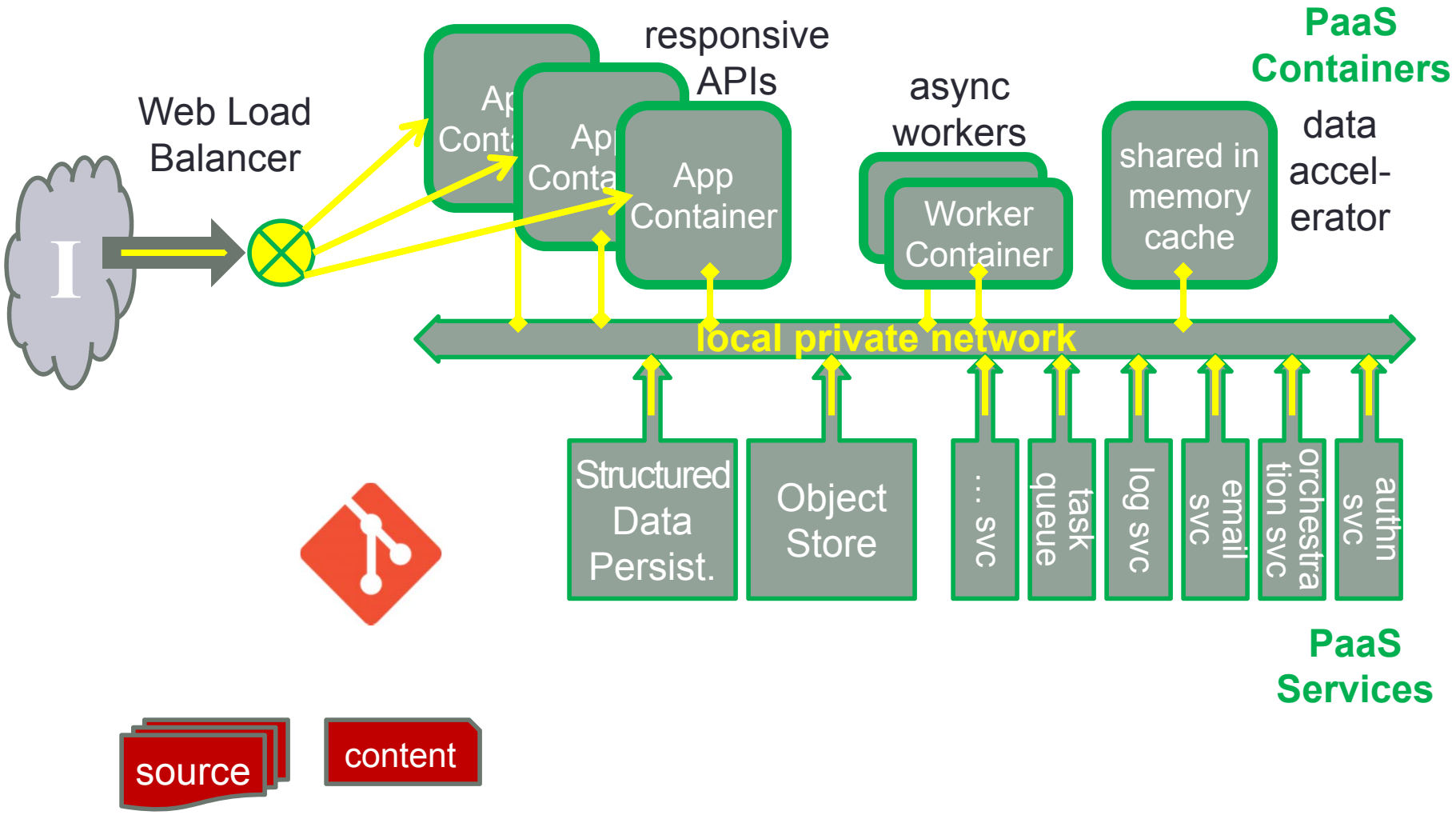
This presentation will describe the experiences, patterns and outcomes of successfully modifying three applications. One application is a large composite providing a multi-tenant service; the second is a complex high performance single tenant application; and the third is an open source multi-tenant application that provides generic UI and community services.

What if your App was simpler?

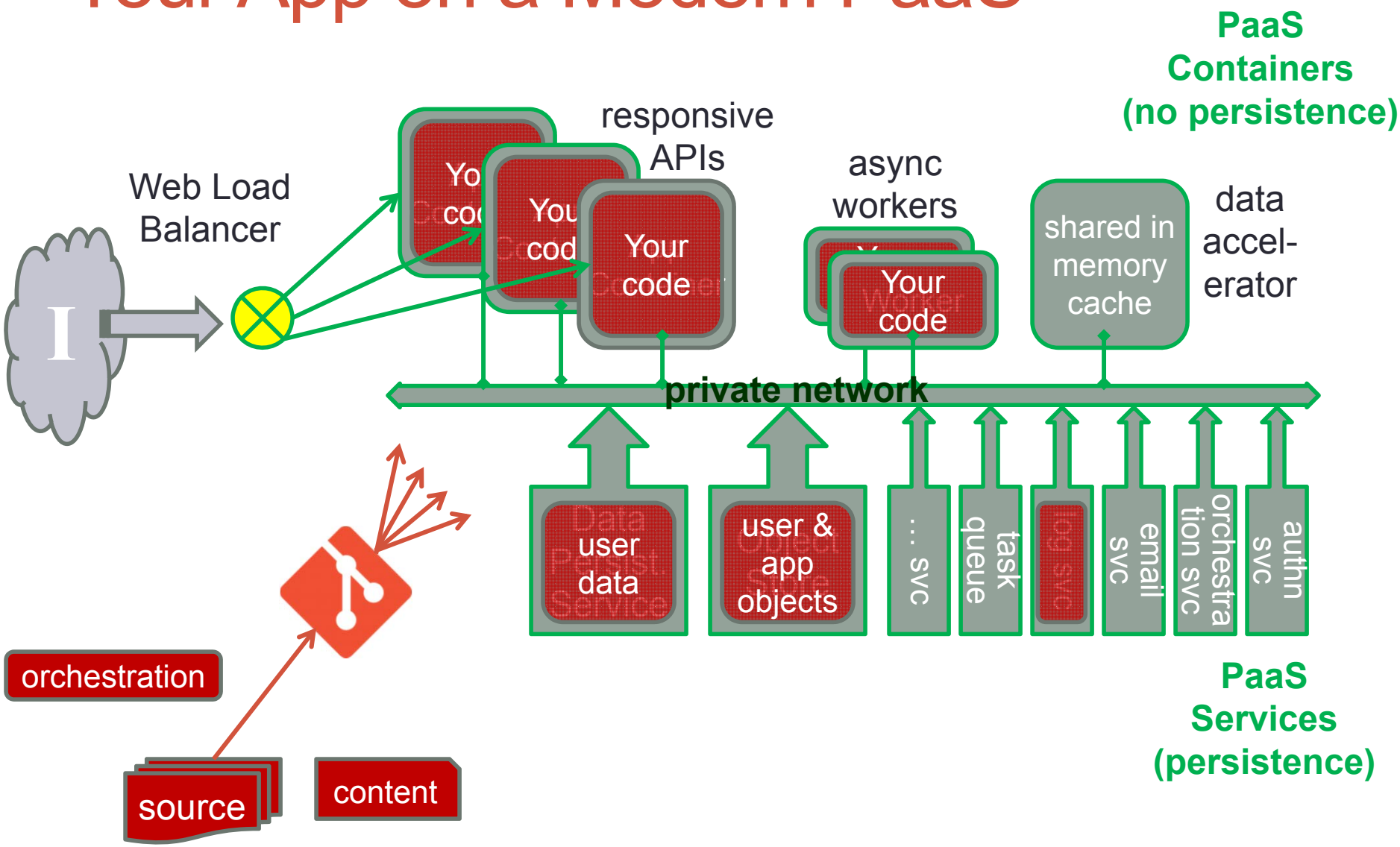


How much faster would you be if…

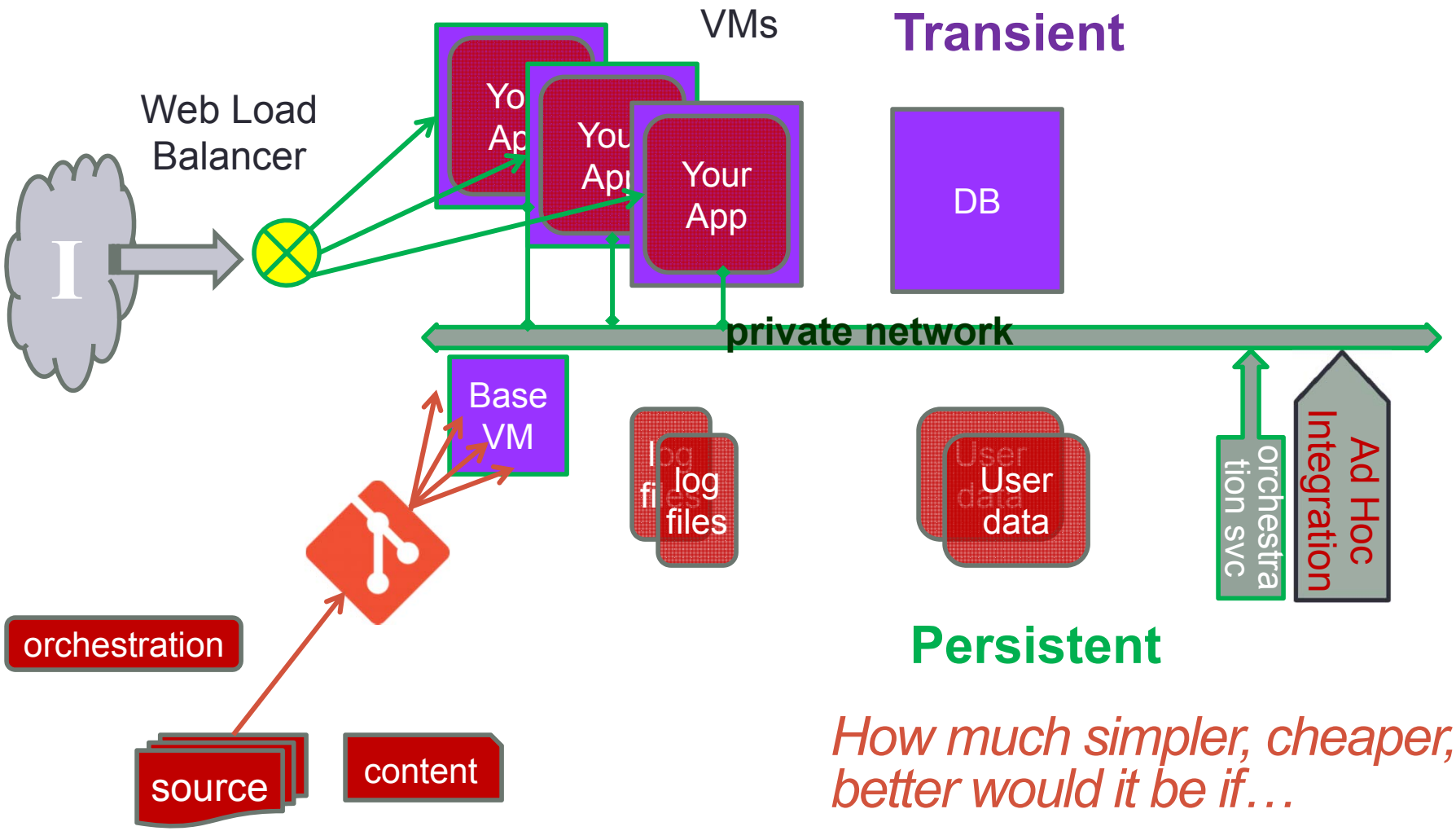
Modern PaaS separates user data, execution and configuration



Your App on a Modern PaaS

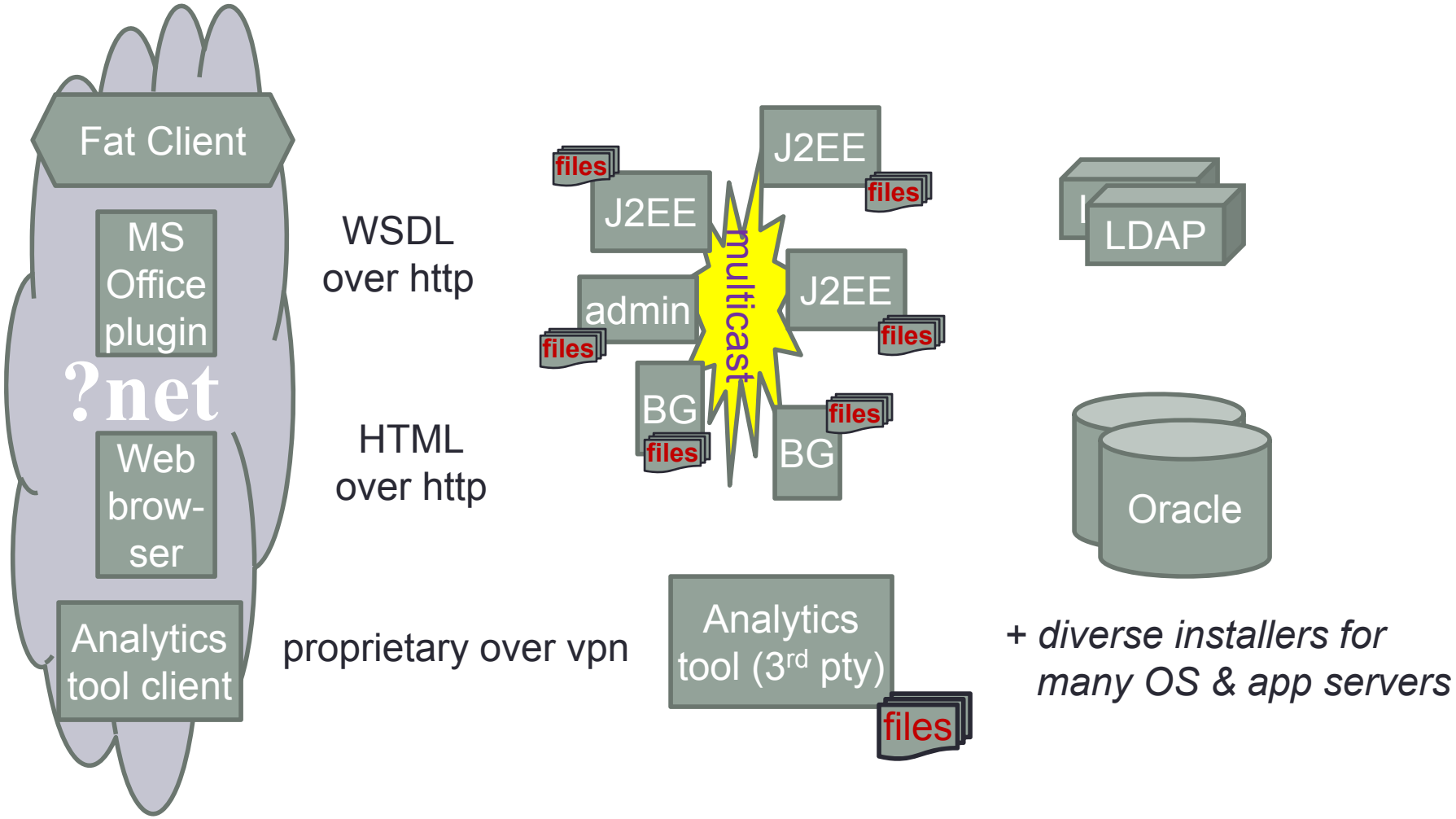


What if?
your App was more like a PaaS app!

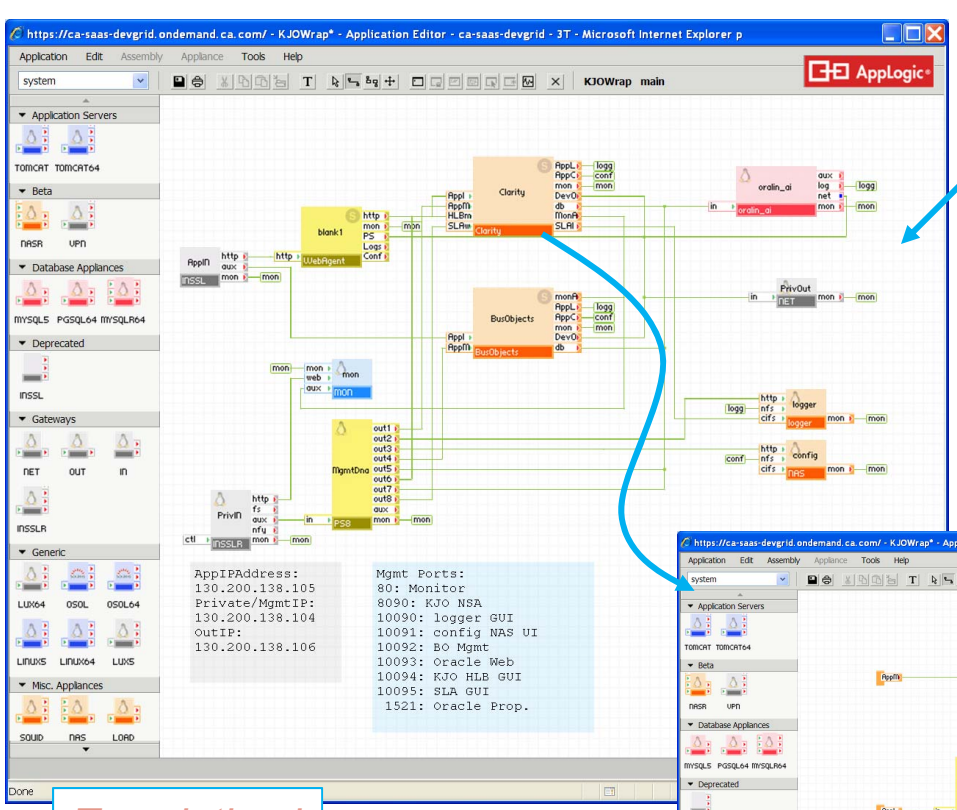


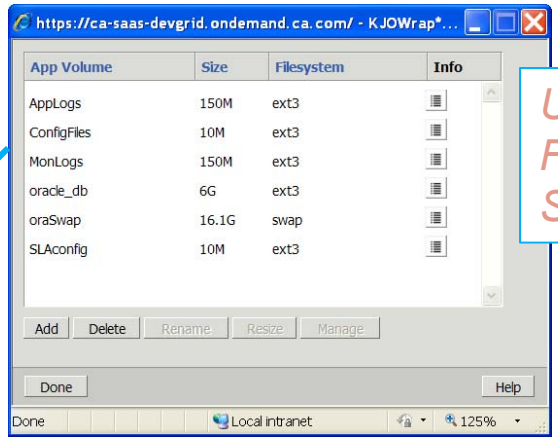
How much simpler, cheaper, better would it be if…

Once upon a time our App looked like this:

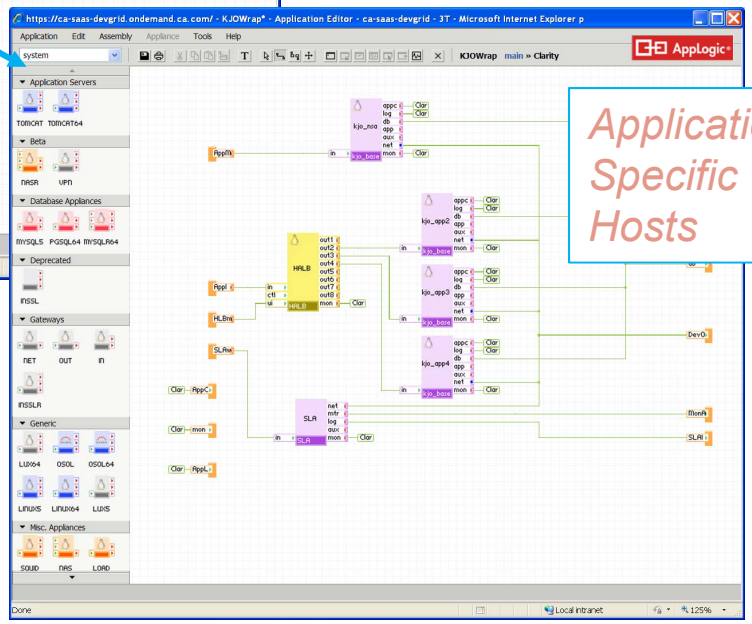


Container Example: Rev. 1 (2011)





Unique File Systems



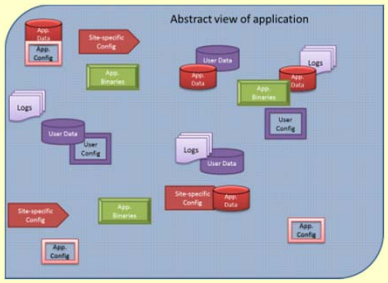
Application Specific Hosts

Templatized Operational Wrapper

"Read only" host images

job #1 is:

separate execution, user data and config



Instance A

=

Common Data

App. Binaries

App. Config

+

User Data

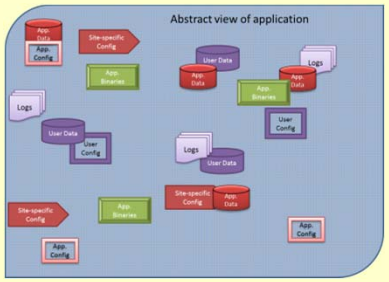
User Config

LOGS

+

Site-specific Config

Re-source Config



New Instance B

= **SAME** + **NEW** + **Transient**

Reversing Traditional Assumptions

Desire super-flexible install

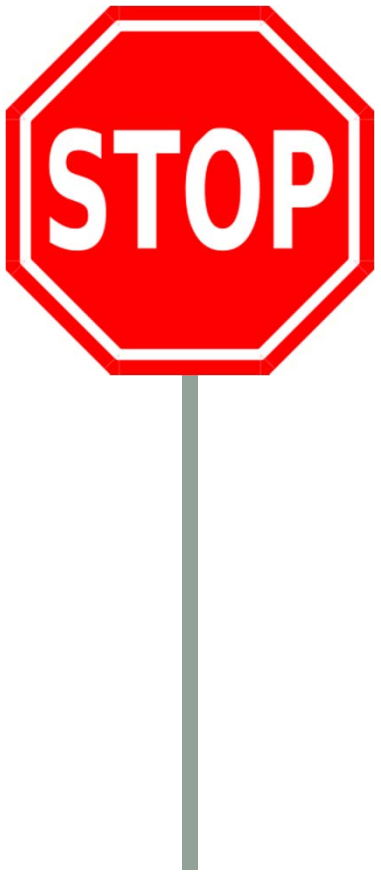
- ☒ Manual, diverse installation & upgrade
- ☒ Idiosyncratic replication & recovery

Presume in-place upgrade

- ☒ Resources, connectivity are static
- ☒ OS version, other elements out of control
- ☒ Owns Filesystem & network ports
- ☒ Tight integrations
- ☒ Intertwined with third party upgrades
- ☒ DR? Scaling?

Automation to fix the problem

- ⛔ Create more complex software to fix software complexity!



Functional goals to improves TCO?

- Backup and restore scenario
- Consistent backup
- User-level Restoration in a new instance
- Flash-cut upgrade
- Disaster Recovery
- Lockdown of executable and its configuration
- …

➔ Reducing manual steps and complexity together improves TCO, TTV, quality, and incrementally moves toward PaaS architecture.

Launch vehicle for job #1

Stage 1: upgrade by rip and replace J2EE apps with off-line database upgrade (extract local config from local files)

Stage 2: upgrade by rip and replace whole maachines. Off-line database upgrade (local config via boundary values, user data on database)

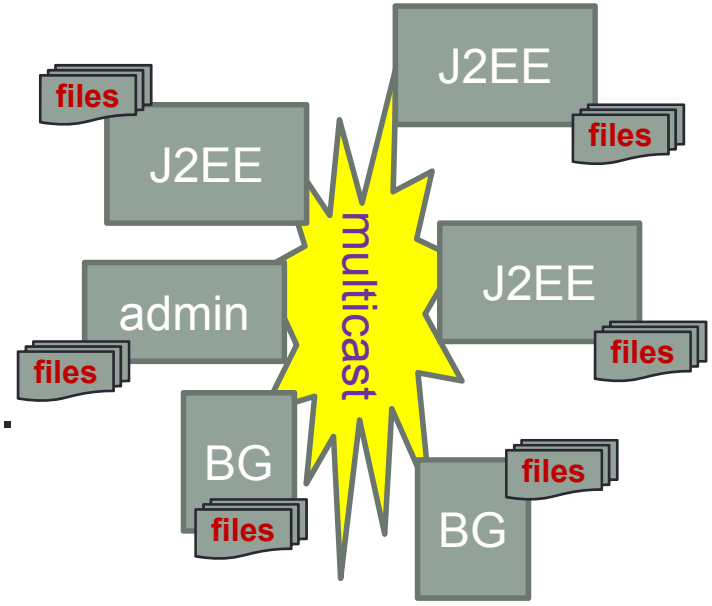
Stage 3: orchestrated deployment based on templates with configuration data

Web App Tier issues

- Logs distributed in executable volumes.
- Config stored in .properties (.war) and other files.
- Options determined by presence/config of plug-ins
- Distinct UI/component for setting/syncing config. No API.
- User and deployment config stored mixed.
- Multicast complexity

Good news:

- User config as data in database ☺
- Same J2EE code in all components.

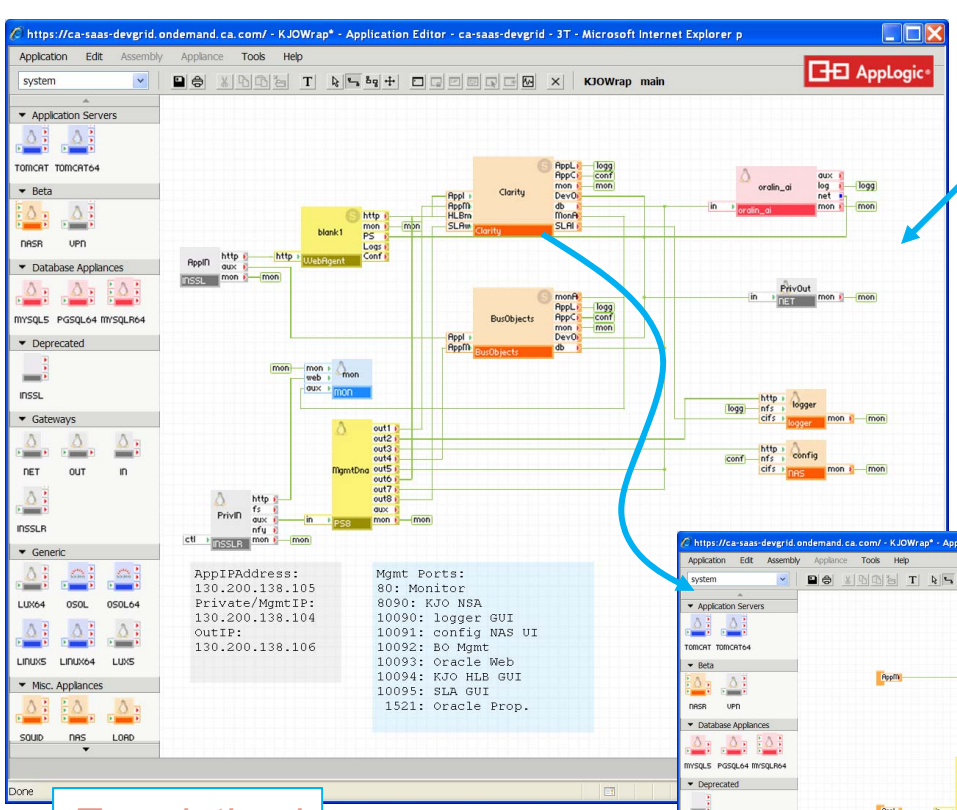


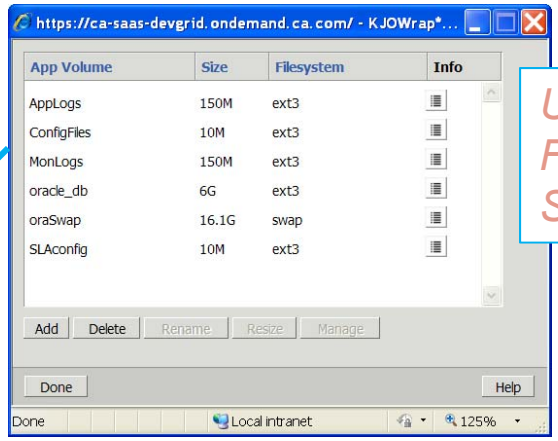
Design Decision
Containers: Baked vrs Built

Desirable :

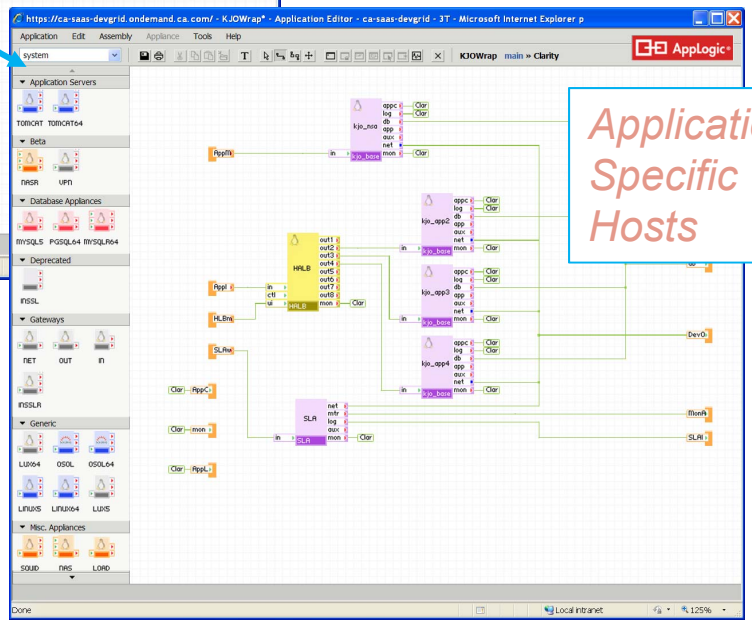
- Automate/centralize logs, manage disk size for logs separate
- Make execution disk (mostly) read-only
- ZDT Upgrade: in place versus replace?
  - Malware detection
  - Vulnerable software
  - App patching

Container Example: Rev. 1 (2011)





Unique File Systems

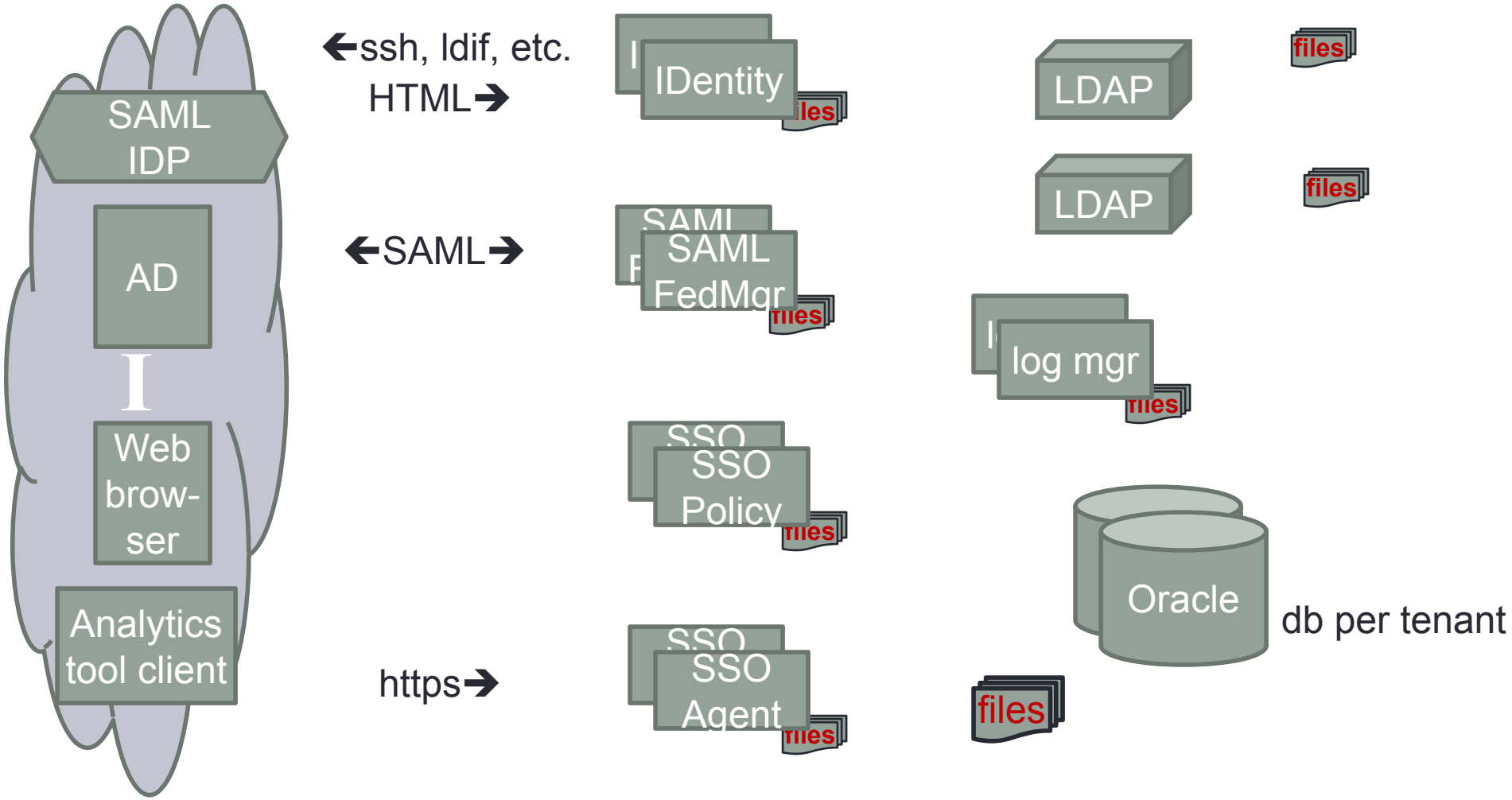


Application Specific Hosts

Templatized Operational Wrapper

"Read only" host images

Evolving our complex *multitenant* App?!



Next Gen Steps

- In general, AuthN should be pulled out of every app, externalized.
- API for base provisioning
- Automation Frameworks
- SDN Network Isolation
- Eradicate backend admin access for config

Process and Cultural Issues

- Establish new end delivery goals first,
  - Resulting process/culture will be “discovered” given the right goals.
  - End goals defined in terms of delivery requirements
  - End goals enforced in terms of concrete acceptance tests
- New disciplines will emerge that are inherently in line with cloud development and a devops mindset.

The cloud is getting more flexible

Your App is becoming more friendly

…and someday they will meet, and they will have a fruitful life together.